U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Processing Image to Geographical Information Systems (PI2GIS) – a learning tool in QGIS

Rui Filipe Canelas Correia

Mestrado em Engenharia Geográfica
Departamento de Geociências, Ambiente e Ordenamento do Território 2017

Orientadora

Professora Doutora Ana Cláudia Teodoro, Professora Auxiliar com Agregação, Faculdade de Ciências Universidade do Porto

Coorientadora

Professora Lia Bárbara Duarte, Professora Auxiliar Convidada, Faculdade de Ciências Universidade do Porto

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____

Acknowledgments

I would like to seek the opportunity to say thank you to all those who supported me during this work. A special thanks goes to my supervisor, Ana Teodoro for the good advises, motivational conversations and scientific support whenever it was needed. Especially mentioned should be Lia Duarte that provided me with an outstanding support in the development of my program.

Furthermore, a big thank you goes to my family and girlfriend, for believing in my potential and for the emotional support.

Abstract

To perform an accurate interpretation of remote sensing images, it is necessary to extract useful information using different image processing techniques. Nowadays, it is usual to use image processing plugins to add new capabilities/functionalities for Geographical Information System (GIS) software. The aim of this work was to develop an open source application to automatically process and classify remote sensing images from a set of satellite input data. The application was integrated in a free open source GIS software (QGIS), automating several image processing steps. It is quick, efficient and easy to use. Furthermore, the use of QGIS for this purpose was justified since it is easy and quick to develop new plugins using Python language. This plugin was inspired in the Semi-Automatic Classification Plugin (SCP) developed by Luca Congedo. SCP allows the supervised classification of remote sensing images, the calculation of vegetation indices such as NDVI (Normalized Difference Vegetation Index) and EVI (Enhanced Vegetation Index) and other image operations. When analysing SCP, it was realised that a set of operations, that are very useful in teaching classes of remote sensing and image processing, were lacking, such as the visualization of histograms, the application of filters, different image corrections, unsupervised classification and other environmental indices. The new set of operations included in the PI2GIS plugin can be divided into three groups: pre-processing, processing, and classification. The application developed was tested in two regions from the North area of Portugal: Aveiro district and Vila Nova de Gaia municipality considering data from Landsat 8 OLI.

Keywords: GIS, QGIS, Semi-Automatic Classification Plugin, vegetation indices.

Resumo

Na área da deteção remota, a interpretação de imagens de satélite é normalmente efetuada com recurso a diferentes técnicas de processamento de imagem de modo a extrair informação interpretável e útil. Atualmente, a utilização/desenvolvimento de *plugins* permite acrescentar novas funcionalidades em softwares livres e de código aberto de Sistemas de Informação Geográfica (SIG) com esse objetivo. O objetivo deste trabalho consistiu no desenvolvimento de uma aplicação *open source* que permite automatizar o processamento e a classificação de imagens de satélite. A aplicação foi implementada no software SIG QGIS. O software QGIS foi escolhido devido à facilidade no desenvolvimento de novos *plugins* através do suporte *online* que fornece, usando a linguagem de programação *Python*. Este *plugin* foi inspirado no "Semi - Automatic Classification Plugin (SCP)" desenvolvido por Luca Congedo, com o qual é possível aplicar diferentes algoritmos de classificação supervisionada, efetuar o cálculo de índices de vegetação tais como NDVI (Normalized Difference Vegetation Index) e o EVI (Enhanced Vegetation Index) e ainda outras operações de processamento de imagem. Ao analisar o SCP, verificou-se que faltavam algumas ferramentas de processamento de imagem que podem ser úteis para serem usadas em contexto de sala de aula, ao nível de ensino superior em aulas relacionadas com o tópico de deteção remota, tais como a visualização de histogramas, a aplicação de filtros, correções nas imagens ao nível de contraste e brilho, algoritmos de classificação não supervisionada e cálculo de outros índices ambientais. O *plugin* - PI2GIS - está dividido em três componentes: pré-processamento, processamento e classificação. O plugin foi testado com imagens Landsat 8 (OLI) em duas áreas do norte de Portugal: no distrito de Aveiro e no concelho de Vila Nova de Gaia.

Palavras – Chave: SIG, QGIS, Semi-Automatic Classification Plugin, índices de vegetação

Index

Figures

Tables

Acronyms

SCP - Semi-Automatic Classification Plugin

Multispectral Scanner (MSS

OLI - Operational Land Imager

ASTER - Advanced Spaceborne Thermal Emission and Reflection Radiometer

MODIS - Moderate Resolution Imaging Spectroradiometer

TM - Thematic Mapp

ETM+ - Enhanced Thematic Mapper Plus

TIRS - Thermal Infrared Sensor

DN – Digital Numbers

CAOP - Carta Administrativa Oficial de Portugal

FCUP – Faculdade de Ciências da Universidade do Porto

QGIS – Quantum GIS

GDAL – Geospatial Data Abstraction Library

GUI - Graphical User Interfaces

API - Application Programming Interface

USGS – United States Geological Survey

WFS - Web Feature Service

NDVI - Normalized Difference Vegetation Index

NDWI - Normalized Difference Water Index

EVI - Enhanced Vegetation Index

MTL - metadata file

DOS1 - Dark Object Subtraction 1

RGB – Red, Green and Blue

EPSG – European Petroleum Survey Group

WGS84 – World Geodetic System 1984

UTM - Universal Transverse Mercator

ETRS89 – European Terrestrial Mercator System 1989

PTTM06 – Portugal Transverse Mercator 2006

1. Introduction

To perform an accurate and realistic interpretation of remote sensing images, it is necessary to extract useful information using different image processing techniques. Nowadays, it becomes usual to use image processing plugins to add new capabilities/functionalities in Geographical Information System (GIS) software.

Remote sensing techniques and GIS are frequently combined in order to extract information of remotely sensed images. For instance, to determine the spatial changes in a specific study area over the years (Land Use Land Cover (LULC)) and/or to estimate environmental indices [1-5]. Proprietary GIS software has been used to extract the spatial information described, such as ArcGIS [1-6]. However, in the recent years open source software, such as System for Automated Geo-Scientific Analyses (SAGA GIS), Geographic Resources Analysis Support System (GRASS GIS) or QGIS, have been increasingly used [7-9] for this purpose.

The application developed in this work was inspired in the Semi-Automatic Classification Plugin (SCP) developed by Luca Congedo [10]. SCP is a free and open source plugin for QGIS software which allows the semi-automatic classification through a supervised classification for remote sensed images, the calculation of vegetation indices such as NDVI (Normalized Difference Vegetation Index) and EVI (Enhanced Vegetation Index) and other image operations, such as automatic conversion to reflectance for different sensors and Region of Interest (ROI) creation. It also processes data from Landsat, Sentinel-2A, Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER) and Moderate Resolution Imaging Spectroradiometer (MODIS).

When analysing SCP, it was realized that a set of image processing operations, that are very useful in remote sensing and image processing teaching classes, were lacking, such as the visualization of histograms, the application of filters, different image corrections, unsupervised classification, and several environmental indices computation, among others.

The main objective of this work was the creation of the Processing Image to Geographical Information Systems - PI2GIS - application inspired on SCP and including a new set of operations. The PI2GIS plugin can be divided into three groups of operations: pre-processing, processing, and classification.

2. State of the art

2.1 Landsat missions

Landsat represents the world's longest continuously acquired collection of space-based moderate-resolution land remote sensing data. Since July 1972, eight missions were launched to produce imagery that provides a unique resource for those who work in agriculture, geology, forestry, regional planning, education, mapping, and global change research. Landsat images are also invaluable for emergency response and disaster relief [11-12]. Figure 1 shows the Landsat missions timeline including the predicted ones.

Figure 1 - Landsat images timeline. Reprinted from USGS website [12]

The Landsat Multispectral Scanner (MSS) sensor was onboard of Landsat 1 through 5 and the images consist of four spectral bands with 60 m of spatial resolution (originally 79 x 57 m, production systems resample the data to 60 m).

Afterwards, Landsat Thematic Mapp (TM) sensor was on board of Landsat 4 and Landsat 5, and the images consist of six spectral bands with a spatial resolution of 30 m for bands 1-5 and 7, and one thermal band (band 6). Then, Landsat Enhanced Thematic Mapper Plus (ETM+) sensor was on board of Landsat 7, and the images consist of seven spectral bands with a spatial resolution of 30 m for bands 1-5, and 7. The resolution for band 8 (panchromatic) is 15 m. Recently, Operational Land Imager (OLI) and Thermal Infrared Sensor (TIRS) are on board of Landsat 8 and consist of nine multispectral bands with 30 m of spatial resolution (bands 1 to 7 and band 9). The

band 8 is the panchromatic band with 15 m of spatial resolution. The thermal 10 and 11 belongs to TIRS with 100 m of spatial resolution [13].

The MTL.txt file (metadata) is included in Landsat 8 Level 1 Data products (Table 1). This file contains information for searching, archiving practices of data and processing data organized in 8 groups of information.

Table 1 – Description of Landsat 8 images metadata file

Group Name	Description
METADATA_FILE_INFO	General information such as: origin, file date and further information
PRODUCT_METADATA	Spacecraft name, geographical extension, path and row numbers, band names and sun's distance and elevation
MIN_MAX_RADIANCE	Maximum and minimum values for Radiance for each band
MIN_MAX_REFLECTANCE	Maximum and minimum values for Reflectance for each band
MIN_MAX_PIXEL_VALUE	Maximum and minimum pixel value for each band, minimum and maximum are always 1 and 65535 because images have 16 bits.
RADIOMETRIC_RESCALING	Rescaling multiplicative and additive for each band useful for Radiance and Reflectance conversions
TIRS_THERMAL_CONSTANTS	Thermal constants useful for TOA Brightness Temperature calculation
PROJECTION_PARAMETERS	Coordinate systems information such as: Ellipsoid, cartographical projection, datum and grid cell size included in all bands.

2.2 Processing image concepts

2.2.1 Histograms

The histograms are usually used to perform an accurate analysis of grayscale images. A histogram is a graphical representation of the quantity of pixels (y axis) in the corresponding value of pixel (x axis). Usually varies between 0 and 255 [14], in 8 bits data. Figure 2 shows an example of a histogram resulted from an image with a red bounding box.

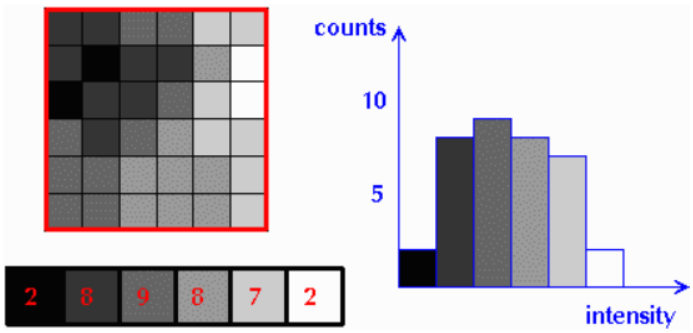


Figure 2 - The procedure for creating an image histogram. Reprinted from National Learning Network for Remote Sensing [14]

2.2.2 Light and Contrast Enhancement

Light and Contrast enhancement operations is a point-by-point method where the result of the operation depends only upon the initial pixel value.

The linear contrast stretch method allows the increasing of brightness, contrast or both by defining the new updated value using the Equation (1).

$$y = cx + b \quad (1)$$

Where *y* is the new pixel value, *x* is the initial pixel value, *c* is the constant that makes the contrast change and *b* is the constant that makes the brightness change.

If the *b* value increases, so will the image brightness. While, if the *b* value decreases, the image brightness will do so as well (Figure 4). On the other hand, by changing the *c* value, the image contrast is increased or decreased.

In the original image's histogram, all values are grouped within a narrow range making any detail impossible to pick up. contrast enhancement operation makes pixel values enlarged so that, the differences are much greater, and therefore features can be easily identified. The histogram associated with the image goes more to the right or left side, respectively (Figure 3).

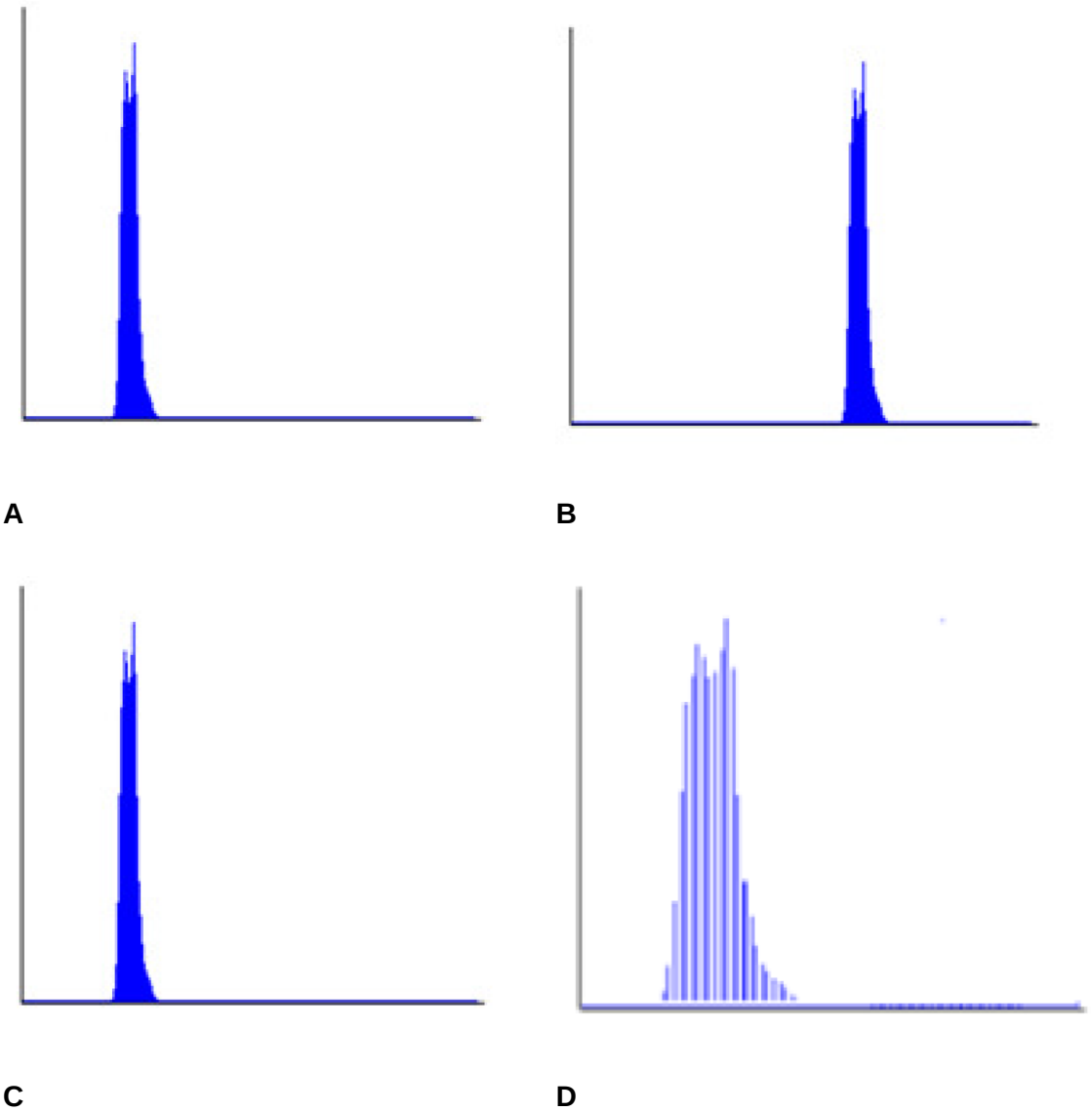


Figure 3 - a) and c) Originals histograms b) histogram with increased brightness d) histogram with increased contrast. Reprinted from National Learning Network for Remote Sensing [14]

2.2.3 Histogram Equalization

The histogram equalization improves the contrast of an image. Histogram equalization is a spatial domain method that produces an output image with uniform distribution of pixel intensity meaning that the histogram of the output image is flattened and extended systematically [15-16].

2.2.4 Filters

The low pass filter allows signals with frequency lower than a certain cut-off to pass. Gaussian filtering allows the smoothing and noise removal. In this kind of filtering kernel matrix is based on the Gaussian distribution, where σ is the standard deviation and μ is assumed to be always 0. Figure 4 shows an example for 1 dimension.

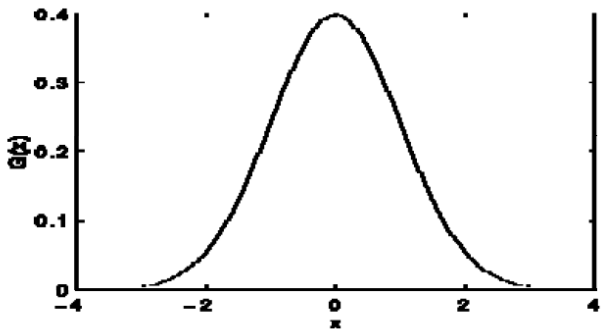


Figure 4 - Gaussian distribution with mean 0 and σ=1. Reprinted from a lecture in Computer Graphics and Image Processing course from The University of Auckland [17]

It is worth noticing that Gaussian kernel coefficients and size depends on the value of σ and larger values of σ produce wider peak and higher the extent of smoothing [17].

Figure 5 shows an example of the effect of the Gaussian filter with different values of σ [18].

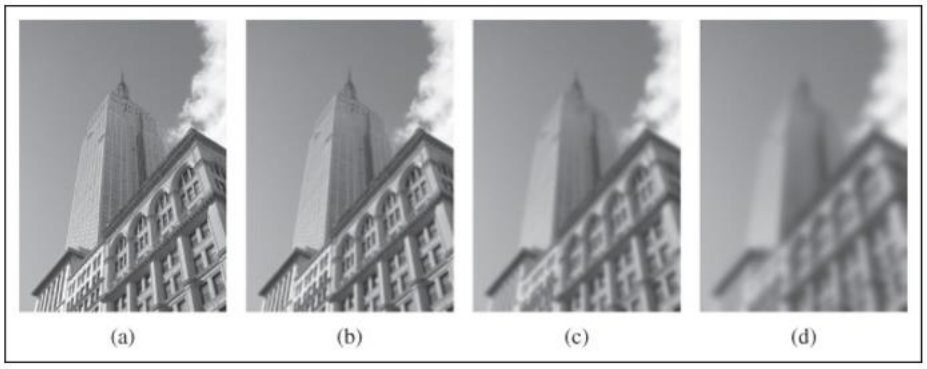


Figure 5 - Gaussian blurring with scipy filters module: (a) – original image, (b) – Filter with σ = 2, (c) – Filter with σ = 5, (d) – Filter with σ = 10. Reprinted from Programming Computer Vision with Python by Jan Erik Solem, [18]

On the other hand, the high pass filters allow high frequencies to pass in the output image, which has the purpose of detecting contours on the image. To apply this filter it is only necessary to invert the low pass Gaussian filter (subtracting original data to the Gaussian low pass filter result). The algorithm of median filtering is based on using a mask with *n* x *n* size (where *n* is odd) that is moved across the image. Then, the new central pixel is defined as the median of pixel values that belong to the mask. Figure 6 shows an example of the application of the median filter [14].

Figure 6 - Example of application of the median filter. Reprinted from National Learning Network for Remote Sensing [14]

2.2.5 Unsupervised classification (k-means algorithm)

K-means is an unsupervised classification algorithm that solves the clustering problem. Clustering is the process of finding a structure in a collection of unlabelled data by organizing objects whose members are similar in some way. K-means is an algorithm which splits the image into different clusters of pixels in the feature space, each of them defined by its center and each pixel is allocated to the nearest cluster [19].

3. Methodology

3.1 PI2GIS application

This work presents a new application composed by a set of functionalities allowing to the creation and visualization of histograms, the application of several filters, different image enhancements corrections, unsupervised classification algorithms and the implementation of several environmental indices. The application was created under QGIS, which is a free open source software, and has its own Application Programming Interface (API) and libraries that can be used in the plugin development, such as QGIS API, PyQt4 API, GDAL/OGR library, and other Python libraries incorporated in the software [20-22]. In order to develop the PI2GIS application, the Python language was used [23]. The graphic interface was created through Qt Designer [24]. Qt Designer is a tool which allows to design and build Graphical User Interfaces (GUI) from Qt components using, for instance, combo boxes, push buttons and labels. The application was composed by the graphic interface file (*ui* file), the *init* script, three dialog scripts and one script of the plugin itself with all the functions for each group.

The graphic interface of the application was created as a toolbar composed by three buttons: *Pre-processing*, *Processing*, and *Classification*. Figure 7 presents the toolbar graphic interface developed.

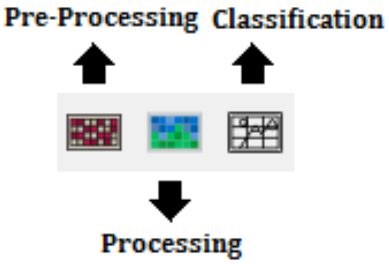


Figure 7 - PI2GIS toolbar

The PI2GIS plugin, considered the three groups defined above, has different goals. The *Pre-processing* group goal is to rescale from 16 – bits to 8 - bits images and to perform several image analysis like histograms, filters and other corrections. In alternative is also possible to convert to Radiance or Reflectance (with or without DOS1 atmospheric correction). The *Processing* group objectives include colour composite, vegetation index calculation and multi-spectral resolution improvement (pan-sharpening). Finally, the *classification* group is intended to perform an unsupervised classification with the colour composition image obtained from the *Processing* group. The Figure 8 shows the PI2GIS plugin workflow with all operations involved in the three groups.

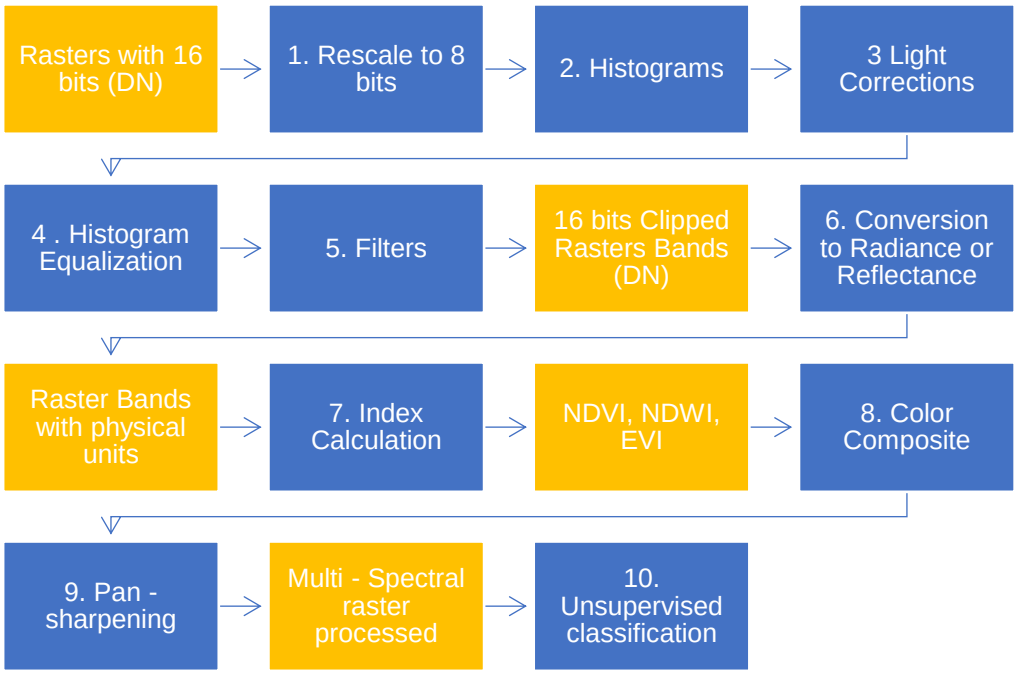


Figure 8 - PI2GIS workflow yellow squares are data and blue squares are processes

3.2 Dataset and study areas

In this work, a Landsat image Operational Land Imager (OLI) was used. Landsat 8 OLI images are delivered in 16 bits in tiff format in Universal Transverse Mercator (UTM) projection Datum World Geodetic System (WGS84) in the 29th zone (EPSG: 32629) coordinate system. The image corresponds to the Landsat path 204 and row 32 and was obtained on the 22nd July 2016, corresponding to summer season, where the vegetation is more active. The same image was used for both study areas Aveiro district and Vila Nova de Gaia municipality.

The Aveiro district is located in west littoral of Portugal (Figure. 1). It is limited by Porto district in the North, Viseu district in the East, Coimbra district in the South and Atlantic Ocean in the West. It covers an area of 2800 km^2. It is characterized by flat terrain, on the majority of the territory above 100 m of altitude, where the coast has about 40 km width in the South. The altitudes increase to East and North extending to Montemuro mountain. The landscape is predominated by Ria de Aveiro and Vouga watershed.

Vila Nova de Gaia municipality is also located in west littoral of Portugal (Figure 9). This municipality belongs to Porto district. It covers an area of 168 km^2 and it is characterized by rugged terrain with 100.12 m as average height and 261 m of altitude as the highest point. The altitudes increase to East and South especially in regions like Monte da Virgem, Serra de Canelas and Olival. The landscape is predominated by urban, industrial and rural regions.

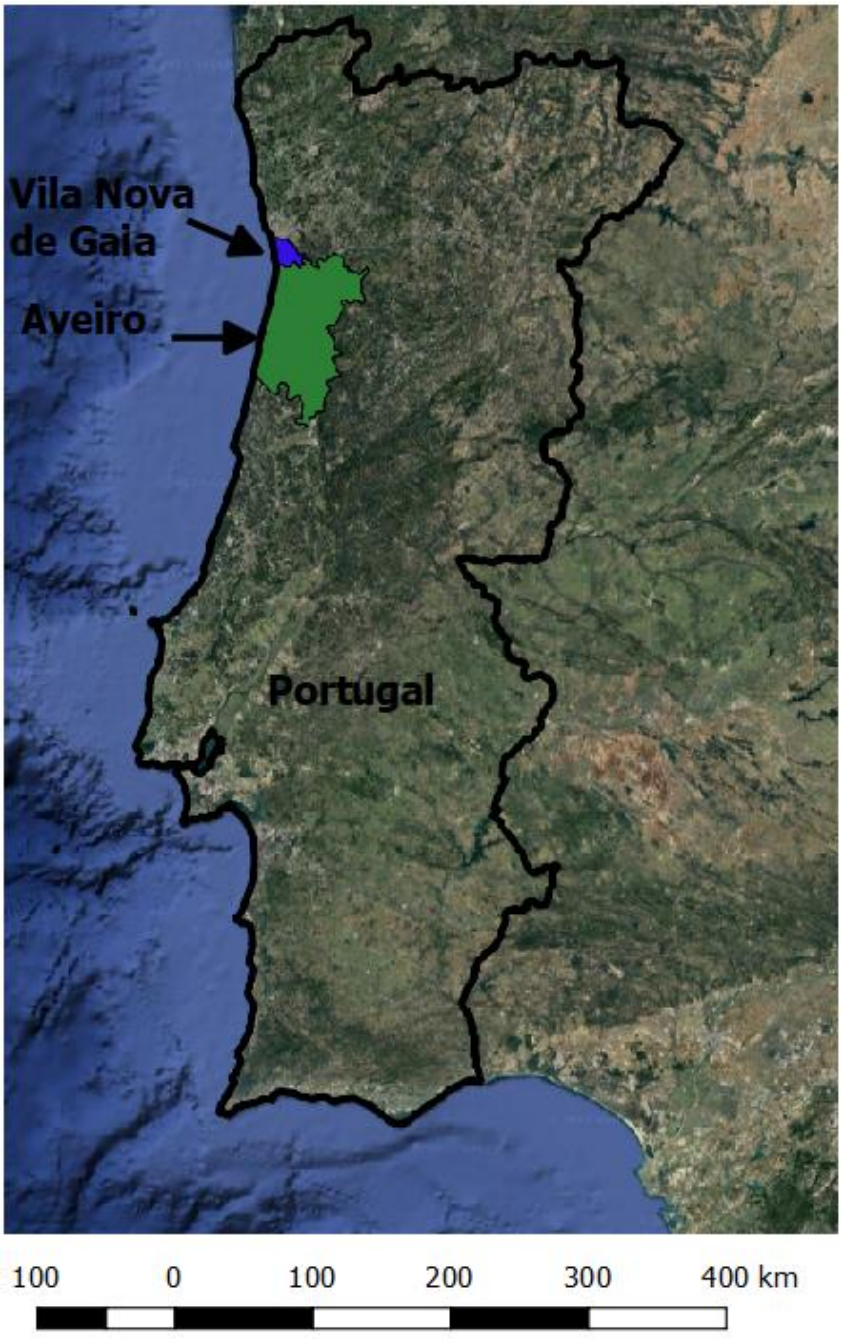


Figure 9 - Aveiro district in green and Vila Nova de Gaia municipality in blue.

3.3 Data Download

Considering the study cases already referred to, Landsat 8 (OLI) images were downloaded using Earth Explorer from the U.S. Geological Survey [25]. A shapefile containing all the Portugal districts “Carta Administrativa Oficial de Portugal” (CAOP)

was downloaded using Web Feature Service (WFS) of Direção – Geral do Território (DGT) [26]. The Aveiro district and Vila Nova de Gaia municipality were extracted from the original shapefile.

3.4 External tools

It is worth noticing that not all tasks were included in the code, and some were considered as external tools. Those external tools were used before or after each part of the plugin in order to prepare data or to correct different kinds of errors. Those tools are usually different versions of *gdalwarp* from GDAL/OGR library to perform coordinate system transformations and clipping. For coordinate system transformations, *gdalwarp* is used with the activation of the parameter *t_srs EPSG:3763* where EPSG is the code associated with the coordinate system used in the study cases. For clipping processes the shapefile with the limits of the study area was used for restricting the region where all the plugin tools should be applied. Furthermore, this clipping process was also important to remove undesirable pixel values around the region of interest by activating the option *srcnodata* to choose the value of the undesirable pixels to be replaced by *nan* selected in *dstnodata* and *crop_to_cutline* and *cutline* options were also applied.

3.5 Treatment of the Data

The shapefiles were projected in the official Portuguese coordinate system PTTM06 – ETRS89 with the GRS80 elipsoid and Transverse Mercator projection (EPSG: 3763). As a result, the Landsat 8 (OLI) images previously defined with (EPSG: 32629) were projected in the same coordinate system as the shapefile (EPSG: 3763) with a coordinate system external tool defined above.

3.6 Pre- Processing group

The Pre-Processing graphic interface was composed by two tabs: *Pre-Processing* and *convertionDN*.

In the Pre-Processing tab the possible distortions of the images can be corrected applying histogram equalisation or filters to Landsat 8 (OLI) images. The Landsat images have 16 – bit format with pixel values varying between 0 and 65535.

To perform the conversion to 8-bits two functions from GDAL/OGR library were implemented. Firstly, *gdal_translate* was used by activating the option *-scale,* secondly *gdalwarp* was applied in order to perform a clip to the study case.

Afterwards, it was possible to create one histogram per band. In order to implement this functionality, the Matplotlib library was used. Matplotlib is a plotting library for Python which is capable of generating histograms [27]. In this plugin, the intensity of values of an input image was read as an array using *np.array*, from *NumPy* library. The array was collapsed into one dimension with the function *np.flatten* so that, it is possible to use the function *hist* that allowed the creation of histograms. Those histograms have the number of pixels in the *y* label and the values in the interval of 0 to 255 in *x* label, so the input has to be 8 - bits images.

To correct different image distortions, a group of three operations were defined as *Light Correction, Histogram Equalization* and *Filters Methods*. Those operations were grouped in two functions: *outputPreview.py* and *pre_operations.py*, respectively. The input of the functions was defined as a combo box widget where is possible to apply the operations to all bands by clicking in the option *All Bands* or just to one band by choosing the right band displayed in the combo box (Figure 10).

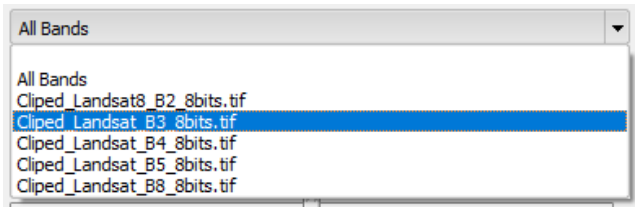


Figure 10 - Combo box example

The function *outputPreview.py* referred previously, allows to apply brightness corrections with the support of the widget *Qslider*. This widget provides a vertical or horizontal slider for controlling a bounded value. It lets the user move a slider handle along a horizontal or vertical groove and translates the handle's position into an integer value [28].

By moving the handle it is possible to increase or decrease the brightness. The interval of values to change brightness correction is from -200 to +200. The results with the corrections are automatically visualized in the QGIS interface through *Preview* button.

In *pre_operations.py* function is possible to apply the *Histogram Equalization function* with the support of *QcheckBox* widget that can be switched on (checked) or off

(unchecked). This widget has the advantage of enable or disables this method without affecting other operations involved in *pre_operations.py* function [29]. The *Histogram Equalization* improves the contrast of images, in this case for 8-bits format. This method flattens the grey level histogram of an image so that all intensities are as equally common as possible and the image intensity becomes normalized. This transform function is, in this case, a *Cumulative Distribution Function* (CDF) of the pixel in the image (normalized to map the range of pixels values to the desire range).

The *np.histogram* function was used to obtain the image histogram and bin edges in float format for 0 to 255 range. The CDF was calculated using the *cumsum()* function from *Numpy* library. In order to normalize the data, the Equation (2) was applied.

$$\frac{255 \times \mathrm{CDF}}{\mathrm{CDF}[-1]} \tag{2}$$

The *np.flatten* function was used to collapse the array in one dimension. Then, it was applied a one-dimensional linear interpolation with the function *np.interp* with data in one-dimension used as input and CDF (normalized data) to obtain the new pixels values [18].

Different types of filters (median, low pass and high pass) were also implemented, with the support of *QcomboBox* and *QspinBox* widgets. *QspinBox* widget allows the user to choose a value by clicking the up/down buttons or pressing up/down on the keyboard to increase/decrease the value currently displayed [30]. It is used to define the size of the footprint for median filter or sigma value for low and high pass filters. In both cases 3×3 is the value set by default. Figure 11 shows the interface.

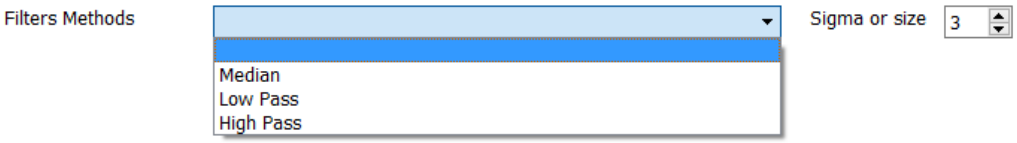


Figure 11 - Interface of filters and sigma or size selection

Median filter is a non-linear filter that allows smoothing the images. This filter is based in a footprint (matrix) typically with size of 3×3 that moves along the image and recalculates the pixels values by taking the pixels median value which is included in that footprint. In this application, the *median_filter* function, from *Scipy* library, was used with the footprint of size 3×3 by default. Low pass filter is a linear filter that also performs smoothing in the image by removing high frequencies keeping the low

frequencies. This filter also uses a footprint (matrix) that moves along the image, but in this case, it is calculated the average value.

It was used *gaussian_filter*, also from *Scipy* library, with sigma = 3 by default (standard deviation for Gaussian Kernel). For contour detection in the image, the high pass filter was used which can be applied by subtracting the original values of pixels, represented in the array, by the values obtained in the low pass filter (described above).

In the *convertionDN* tab (with *convertionDN.py* associated) is intended to perform a conversion of the bands from Digital Numbers (DN) to radiance or reflectance with or without atmospheric corrections for images with 16 – bits, so the multispectral bands are prepared to be processed. In this function, it is used a cycle *for* in all images in a specified directory. It is required to have Landsat 8 (OLI) bands (2 to 8) as an input, since those are the most relevant bands for the processing. This function was inspired on a code obtained from *GitHub*, which uses *GDAL* and *Numpy* libraries [31] and based on Landsat 8 Data Users Handbook [32]. The effects of the atmosphere must be considered to obtain the reflectance at the surface. Therefore, Dark Object Subtraction 1 (DOS1) atmospheric correction algorithm was chosen to improve the estimation of land surface reflectance [33]. The code implemented was based on SCP manual [10].

Before the equations implemented in the *convertionDN* tab, 16 – bits format for the input images were required so, a clipping external tool was applied for the same shapefile of the district of municipality used in the treatment of data. Once the Landsat 8 OLI images have a black outlier with pixel values 0, this clipping process was used by activating the option responsible form replacing pixel values with 0 to “nan” maintaining the same 16 – bits format.

To perform the conversion described above, it was necessary to read the metadata file (MTL), a metadata file in *.txt* format:

- RADIANCE_MULT_BAND (Radiance Multiplicative rescaling factor)
- RADIANCE_ADD_BAND (Radiance Additive rescaling factor)
- RADIANCE_MAXIMUM_BAND
- REFLECTANCE_MULT_BAND (Reflectance Multiplicative rescaling factor)
- REFLECTANCE_ADD_BAND (Reflectance Additive rescaling factor)
- REFLECTANCE_MAXIMUM_BAND

- SUN_ELEVATION (Solar Elevation Angle)
- EARTH_SUN_DISTANCE

In order to read the data available in the MTL file, a dictionary was created, which is a group of key-value elements, where the key was the number of the Landsat 8 band and the value was the relevant variables for the conversions. For instance, in the radiance additive rescaling factor in the band 1 (RADIANCE_ADD_BAND_1 = -60.75992), it is created a key with the number 1 and the value 60.75992. This dictionary is created with the support of the function *line.slipt.*

Conversion to Radiance

The spectral radiance at the sensor's aperture (L_λ), measured in watt per steradian (square radian) per square meter per micrometre (W $\mathrm{ster}^{-1}m^{-2}m^{-6}$) was calculated with the support of radiance rescaling multiplicative and additive factors in the Equation (3):

$$L_\lambda = M_L\, Q_{cal} + A_L \quad (2)$$

Where M_L is the band – specific multiplicative rescaling factor from Landsat 8 metadata (RADIANCE_MULT_BAND_x, where x is the band number), A_L is the band-specific additive rescaling factor from Landsat 8 metadata (RADIANCE_ADD_BAND_x, where x is the band number) and Q_{cal} is the quantized and calibrated standard product pixel values (DN).

Conversion to Reflectance TOA

Using the same procedure, the 16-bit integer values can be converted to Top of Atmosphere (TOA) Reflectance, without correction for solar angle ($\rho_{\lambda'}$ dimensionless) (Equation 4):

$$\rho_{\lambda'} = M_p\, Q_{cal} + A_p \quad (3)$$

Where M_p is the reflectance multiplicative rescaling factor from Landsat 8 metadata (REFLECTANCE_MULT_BAND_x, where x is the band number), A_p is the reflectance additive rescaling factor from Landsat 8 metadata (REFLECTANCE_ADD_BAND_x from the where x is the band number) and Q_{cal} is the quantized and calibrated standard product pixel values (DN).

To perform the solar angle correction, it is required to use the Equation (5):

$$\rho_{\lambda} = \frac{\rho_{\lambda'}}{\sin(\theta)} \quad (5)$$

Where ρ_{λ} is the TOA Planetary Reflectance with solar angle correction and θ is the solar elevation angle obtained from metadata.

It is worth noticing that this is a one-step conversion by using reflectance multiplicative and additive factors available in metadata file. On the other hand, Landsat TM and ETM+ are rescaled in 8-bit (DN), so TOA reflectance is obtained through a two – step process: conversion to radiance values using the BIAS and GAIN values which are available specifically in Landsat 7 metadata file [34].

Dark Object Subtraction 1 (DOS1)

The DOS1 is an image-based atmospheric correction defined by Chavez as the "basic assumption is that within the image some pixels are in complete shadow and their radiances received at the satellite are due to atmospheric scattering (path radiance)" [33]. In this plugin, DOS 1 was applied based on the equations given by Congedo [10].

DOS 1 radiance correction

The path radiance L_p given by Sobrino [35] is the radiance resulted from the interaction of the electromagnetic radiance with the atmospheric components (molecules and aerosols) that can be obtained according to Equation (6):

$$L_p = L_{min} - L_{1\%} \quad (6)$$

Where L_{min} is the radiance that corresponds to a digital count value for which the sum of all the pixels with digital counts lower or equal to this value is equal to the 0.01% of all the pixels from the image considered [35] and $L_{1\%}$ is the radiance of the dark object, assumed to have a reflectance value of 0.01. In Landsat images, the L_{min} is calculated with the equation (7):

$$L_{min} = M_L\, DN_{min} + A_L \quad (7)$$

Where M_L is the band – specific multiplicative rescaling factor from Landsat 8 metadata (RADIANCE_MULT_BAND_x, where x is the band number), A_L is the band-specific additive rescaling factor from Landsat 8 metadata (RADIANCE_ADD_BAND_x, where x is the band number) and DN_{min} is the minimum DN value for each 16 bits band

considered. It was used the function *np.amin* from *Numpy* library to calculate the lowest value of a certain array of values.

To calculate the radiance of the dark object $L_{1\%}$ given by Equation (9), it is necessary to calculate solar exoatmospheric spectral irradiances (ESUN) measured in watt per square meter per micrometre (W $m^{-2}m^{-6}$) [36] given by:

$$ESUN_{\lambda} = (\pi * d^2) * \frac{M'_L}{M'_p} \quad (8)$$

Where: M'_L is the RADIANCE_MAXIMUM_BAND_x, available in the Landsat 8 metadata (where x is the band number), M'_L is the REFLECTANCE_MAXIMUM_BAND_x, available in the Landsat 8 metadata (where x is the band number) and d Earth-Sun distance in astronomical units available in the Landsat 8 metadata as (EARTH_SUN_DISTANCE).

Then, the radiance of the dark object $L_{1\%}$ is calculated as given in Equation (9):

$$L_{1\%} = 0.01\,[(ESUN_{\lambda} \cos\theta_s\, T_z) + E_{down}] * \frac{T_v}{\pi * d^2} \quad (9)$$

Where T_z is the atmospheric transmittance in the viewing direction, T_v is the atmospheric transmittance in the illumination direction, E_{down} is the downwelling diffuse irradiance, $ESUN$ solar exoatmospheric spectral irradiances calculated with the equation (8).

So, from the Equation (6) is obtained by combining (7) and (9):

$$L_p = M_L\, DN_{min} + A_L - 0.01\,[(ESUN_{\lambda} \cos\theta_s\, T_z) + E_{down}] * \frac{T_z}{\pi * d^2} \quad (10)$$

Although there are several DOS techniques (e.g. DOS1, DOS2, DOS3, DOS4), based on different assumption about T_v, T_z, and E_{down} [35]. This plugin is intended to make an image-based atmospheric correction which means, that no atmospheric conditions information is needed. So, the easiest technique used was DOS1, where the following assumptions are made: T_v = 1, T_z = 1 and E_{down} = 0.

Thus, path radiance is given by Equation (11):

$$L_p = M_L\, DN_{min} + A_L - \frac{0.01\,(ESUN_{\lambda} \cos\theta_s)}{\pi * d^2} \quad (11)$$

DOS 1 reflectance correction

The resulting surface reflectance is given by Equation (12):

$$\rho = \frac{\pi * \left(L_\lambda - L_p\right) * d^2}{ESUN_\lambda \cos\theta_s} \quad (12)$$

After applying the previous equations to obtain reflectance with DOS1 correction two problems were verified: (1) wrong coordinate system; (2) and the presence of a black outlier with all pixels having value 0.01 around the district of interest. To solve the wrong coordinate system, the external tool for coordinate systems transformation was applied and PTTM06 – ETRS89 was defined. Afterwards, the external tool for clipping was applied by selecting the value 0.01 and replacing by nan.

3.7 Processing group

Through the *Processing* button it is possible to perform a colour composite scene, followed by pan-sharpening process to improve multispectral image resolution. It is also possible to calculate different environmental indices with special focus on vegetation and water indices such as NDVI, EVI and Normalized Difference Water Index (NDWI). In the graphic interface, a combo box was created with four options: *Color composite*, *NDVI*, *NDWI* and *EVI*. In order to create the colour composite, *gdalogr:merge*, algorithm from GDAL library, was used.

As a result of this process, a black outlier with all pixels having the value 0 was created around the region of interest. So, to solve the problem, a clipping process external tool was applied by replacing the undesirable values from 0 to nan.

The vegetation indices were implemented through *gdalogr:rastercalculator*, algorithm from *GDAL library*, defining the Equations 13-15:

$$NDVI = \frac{\mathrm{NIR} - \mathrm{RED}}{\mathrm{NIR} + \mathrm{RED}} \quad (13)$$

$$EVI = G\,\frac{NIR - RED}{NIR + C1RED - C2BLUE + L} \quad (14)$$

$$NDWI = \frac{GREEN - NIR}{GREEN + NIR} \quad (45)$$

Where NIR, RED, BLUE and GREEN values are the near-infrared reflectance, red reflectance surface, blue reflectance surface and green reflectance surface, respectively [37,38]. L (L=1) is a canopy background adjustment term, and C1 (C1=6) and C2 (C2=7.5) are the coefficients of the aerosol resistance term and G is a gain or scale factor (G=2.5) [39].

To perform the pan-sharpening, two tools from *OrfeoToolbox*, were implemented in a sequence: *Superimpose* sensor and *Pansharpening Ratio Component Substitution* (RCS). The multispectral bands (MS) obtained using colour composite tool with 30 m of spatial resolution and the panchromatic band (PAN) with 15 m of spatial resolution were used as an input. The superimpose tool performs the projection of an image into the geometry of another one [40] as a result, the MS image has the same extension as the PAN image.

Afterwards, *Pansharpening* tool with the RCS algorithm was implemented. Pansharpening is a process of merging the image PAN and lower resolution MS image to create a single high-resolution colour image [41].

The high – resolution colour image was obtained with a black outlier with pixels values “0” created around the district or municipality as a result of the Orfeo ToolBox. So, a clipping external tool was applied by replacing the pixel values from 0 to nan.

3.8 Classification group

The *Classification* button allows to introduce the unsupervised classification algorithm which has the advantage of not requiring the definition of training classes before running the algorithm since each class is defined only according it’s spectral properties. In this first version, the application incorporates the K-means algorithm. This functionality is also an innovation of PI2GIS application, when compared to SCP.

In this application, the algorithm was applied based on *OrfeoToolbox* library [42]. In the classification group, a multispectral image was used as an input and a set of parameters were defined such as: training set size (value 100 by default), convergence threshold (value 0.0001 by default), maximum number of iterations (value 1000 by default) and the number of classes. In order to support the definition of those parameters the widgets *QspinBox* and *QdoubleSpinBox* were used. The widget

QspinBox is designed to handle integers and discrete sets of values, while *QdoubleSpinBox* is designed to handle floating point numbers [29]. The widget *QdoubleSpinBox* was used for the convergence threshold and the remaining parameters were supported by *QspinBox* as presented in FFigure 12.

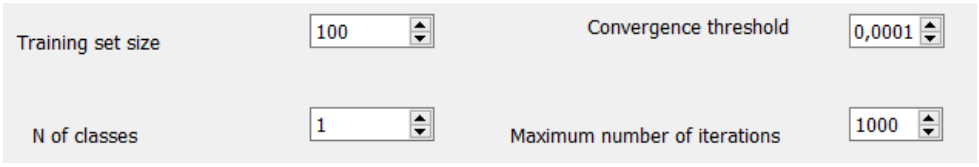


Figure 12 - Parameters to be selected in Classification group

The input image is the resulting multispectral high-resolution image from the pan sharpening process with an invisible outlier with *Nodata* values around the clipped region. As a result of the unsupervised classification, several classes were defined according it´s spectral properties and *Nodata* values were also considered to be a class.

To remove the classified values that were *Nodata* before, a clipping operation was implemented in the code with the support of *gdalwarp* that was included in the classification group by selecting the shapefile and *crop_to_cutline* and cutline options were activated.

A spin box was created to choose which class (source data in *gdalwarp)* should be removed in the previous classified image and switched again to *nan* (destine data in *gdalwarp).* Figure 13 shows the interface of how the inconvenient pixels values are selected.

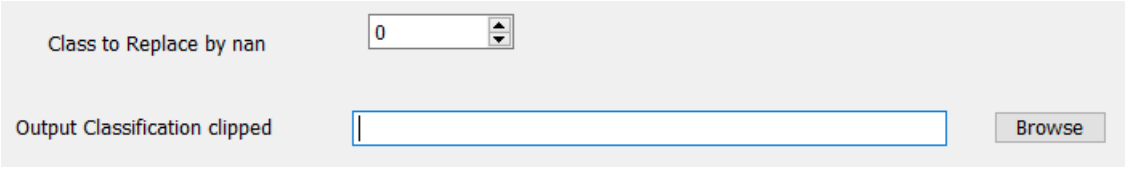


Figure 13 - Interface of class removal value

4. Results and Discussion

4.1 Aveiro district

4.1.1 Pre – Processing results for Aveiro district

In order to test the pre-processing group, the band 2 of Landsat 8 OLI image was selected. In the *Pre-Processing* group, the images are rescaled from 16-bits to 8-bits. Then, the histograms for each band were created. Figure 14 shows the histogram of band 2.

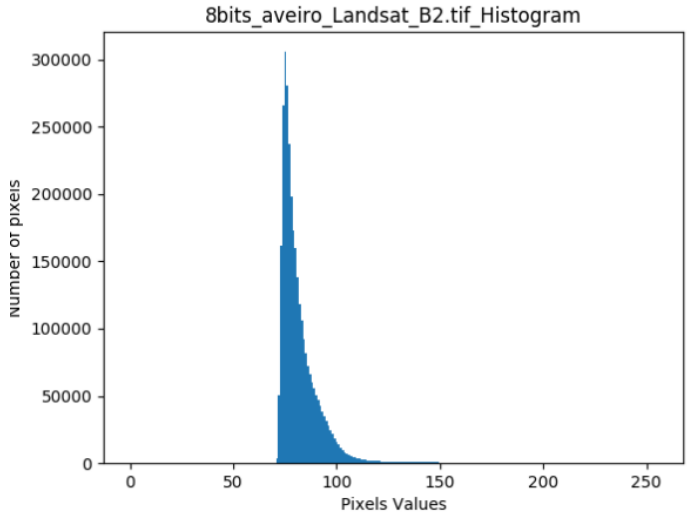
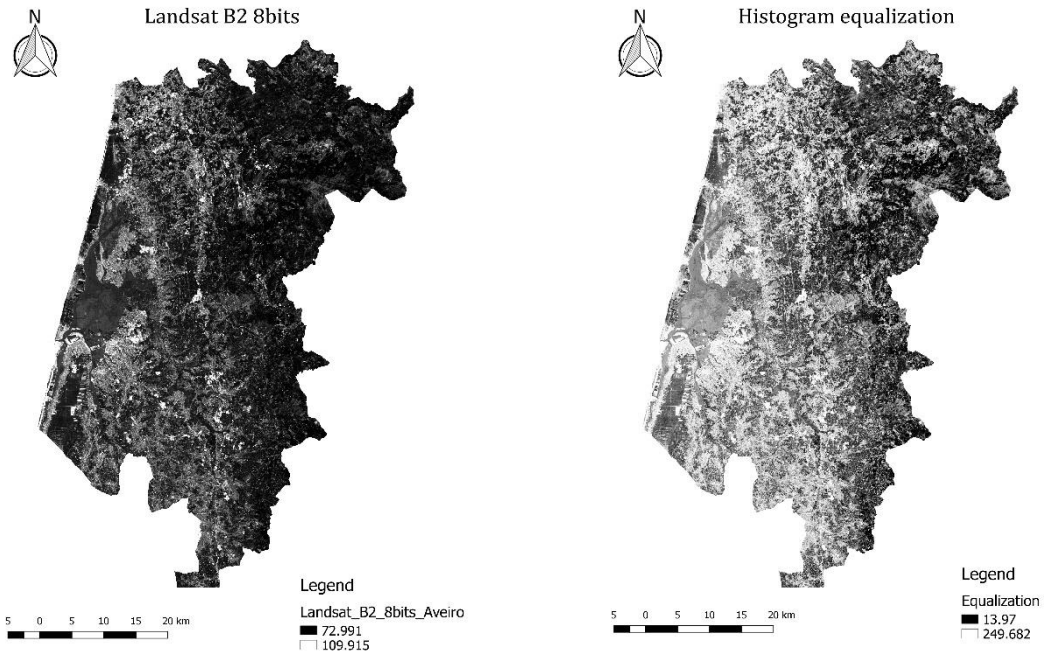


Figure 14 - Histogram for band 2

From the result obtained it was concluded that the image has generally low contrast and brightness, so it can have an improvement in contrast applying the histogram equalisation. Figure 15 presents the comparison between the original 8-bits image (on the left side) with the image resulted in the histogram equalization.



A **B**

Figure 15 - a) Landsat B2 8 bits; b) Histogram equalization of Landsat B2 8 bits.

The histogram equalization is a process that increases the contrast of images as it is possible to verify in the Figure 16. Furthermore, the Figure 16 shows the histogram of the resulting image from the equalization process.

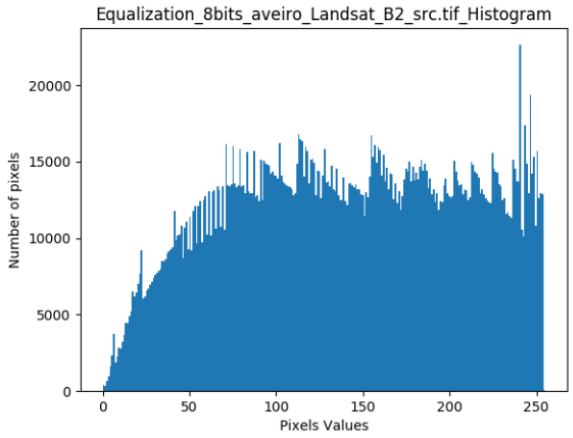
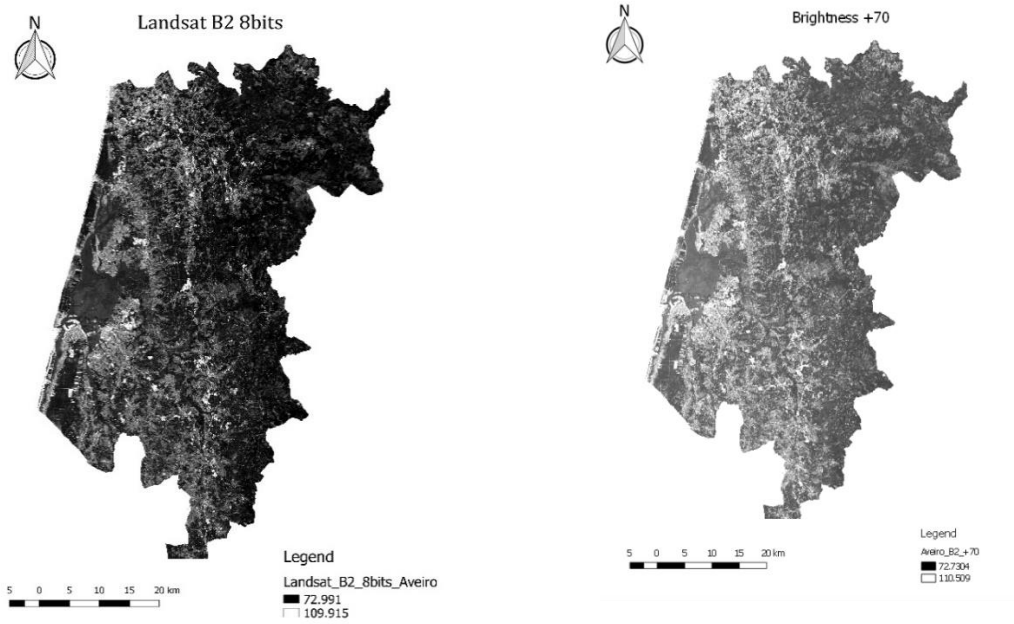


Figure 16 - Histograms resulted from the histogram equalization process

By analysing the Figure 16 is possible to verify an increase in contrast because the histogram shows greater differences between pixel values and therefore the features in

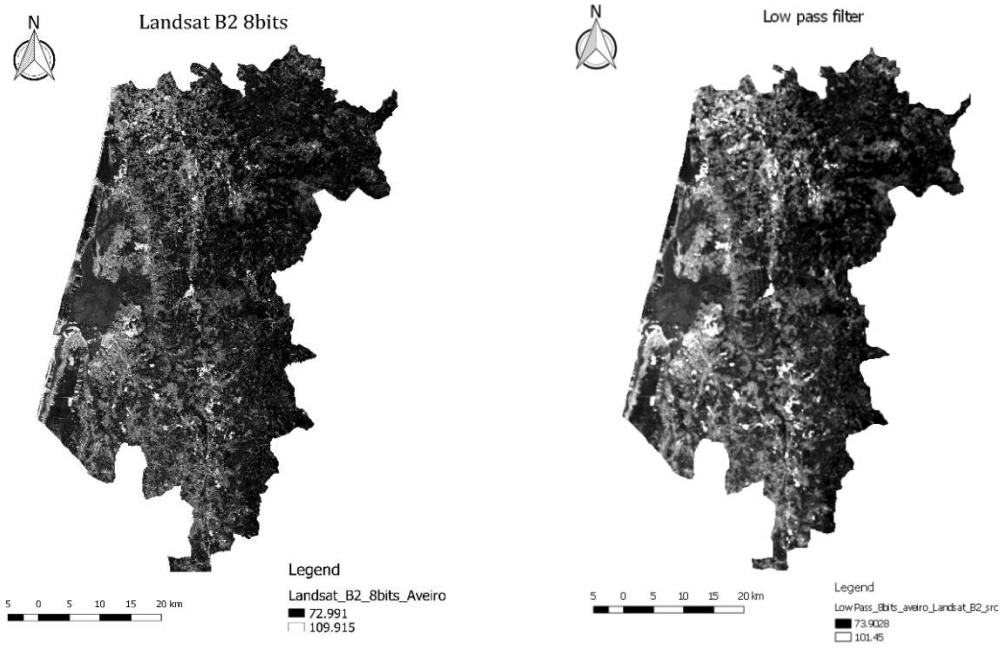
the image are easily identified as is possible to verify in the Figure 15b). PI2GIS also allows manipulating the brightness effect in in the grayscale band 2 image. The Figure 17 shows the difference in the brightness by applying an increase of 70 to each value of pixel.



a **b**

Figure 17 - a) Landsat B2 8 bits; b) Landsat B2 8 bits with increased brightness

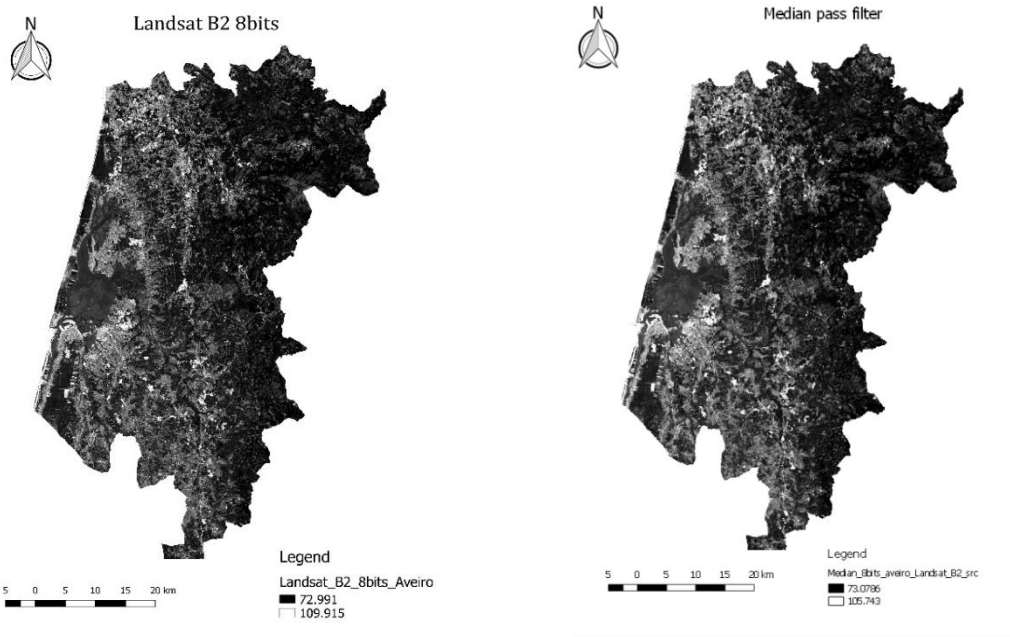
Low pass filter allows low frequencies to be maintained. This effect is easily verified with an increase of the smoothing effect in the Figure 18 which presents a comparison between the original 8-bits image (on the left side) with the image resulted by the low pass filter (on the right side).



A **B**

Figure 18 - a) Landsat B2 8 bits; b) Landsat B2 8 bits with low pass filter applied

Afterwards, median filter was applied and a smoothing is also expected. The Figure 19 presents the comparison between the original 8-bits image (on the left side) with the image resulted from the median filter.



A **B**

Figure 19 - a) Landsat B2 8 bits; b) Landsat B2 8 bits with median filter applied

By analysing the Figure 19, is possible to verify that the smoothing effect is not so present by comparing with the Figure 18. To verify the smoothing effect with more detail a zoom was performed and presented in the Figure 20.

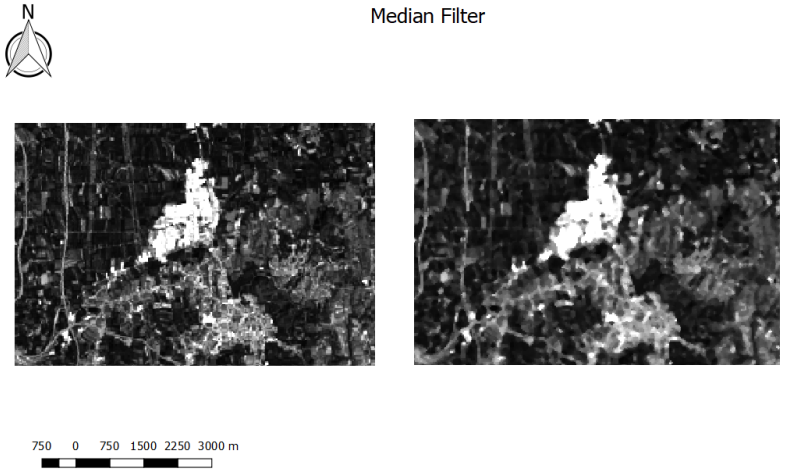
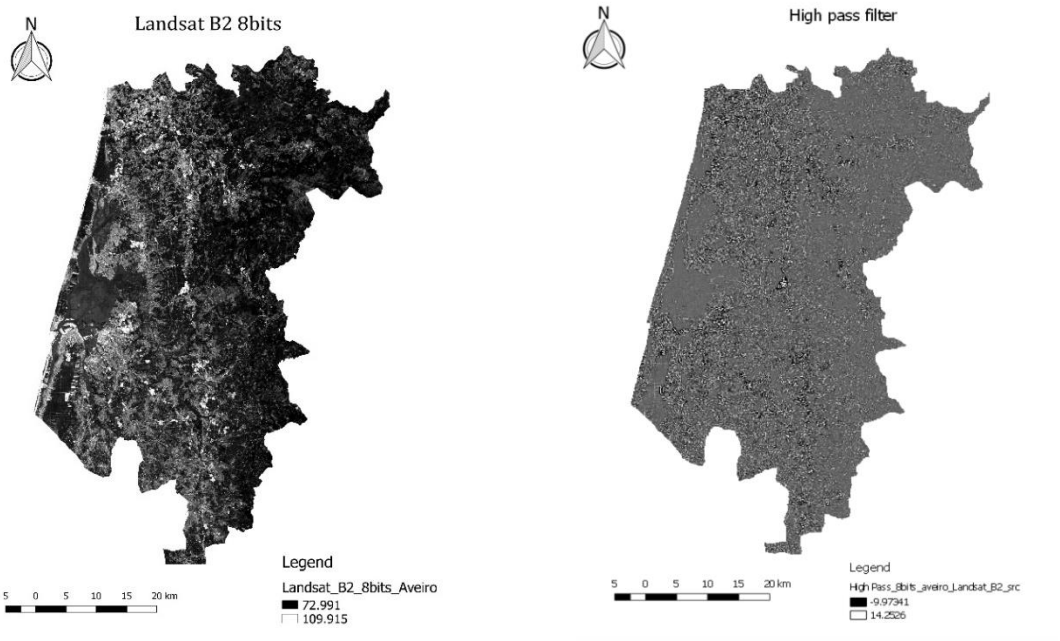


Figure 20 - Zoom in on the image before and after the median filter have been applied

Finally, with a different propose of the previous filters, the high pass filter is used in the contour detection. The Figure 21 presents the comparison between the original 8-bits image (on the left side) with the resulting image from high pass filter with contours of buildings and another features present in Aveiro.



A **B**

Figure 21 - a) Landsat B2 8 bits; b) Landsat B2 8 bits with High pass filter applied

Furthermore, the Figure 22 shows a zoom of the previous Figure where a contour detection is easily identifiable.

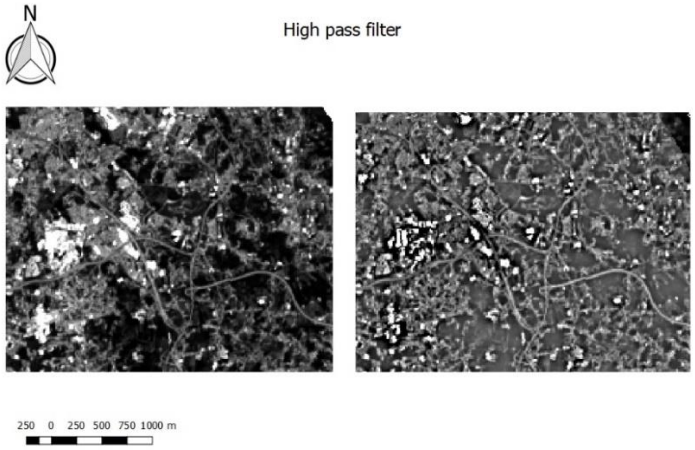


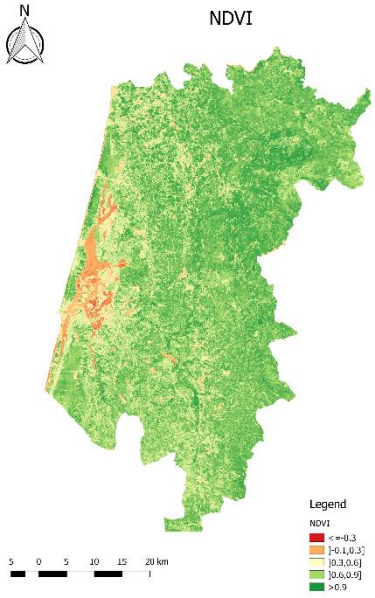
Figure 22 - Zoom in on the image before and after the High pass filter have been applied

By analysing the Figure 22 is possible to verify that the houses and buildings are enhanced.

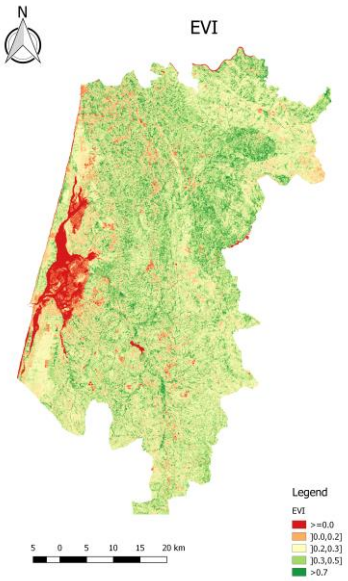
4.1.2 Processing group for Aveiro district

In the processing group, RGB Composite, NDVI, EVI and NDWI indices were obtained. Figure 23 presents the results obtained.

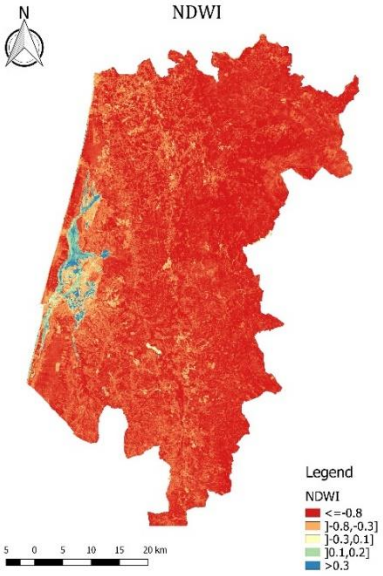
A



B



C



D

Figure 23 - a) RGB composite; b) NDVI map; c) EVI map; d) NDWI map

NDVI and EVI vegetation indices presented higher values in the spring and summer season, as the image was obtained in July 2016. EVI presented lower values compared to NDVI, as EVI corrects for some distortions in the reflected light caused by the particles in the air as well as the ground cover below the vegetation [43]. Through NDWI map it is possible to clearly identify the water bodies.

After obtaining the indices, the pan sharpening functionality was tested. Figure 24 presents the difference before and after pan-sharpening.

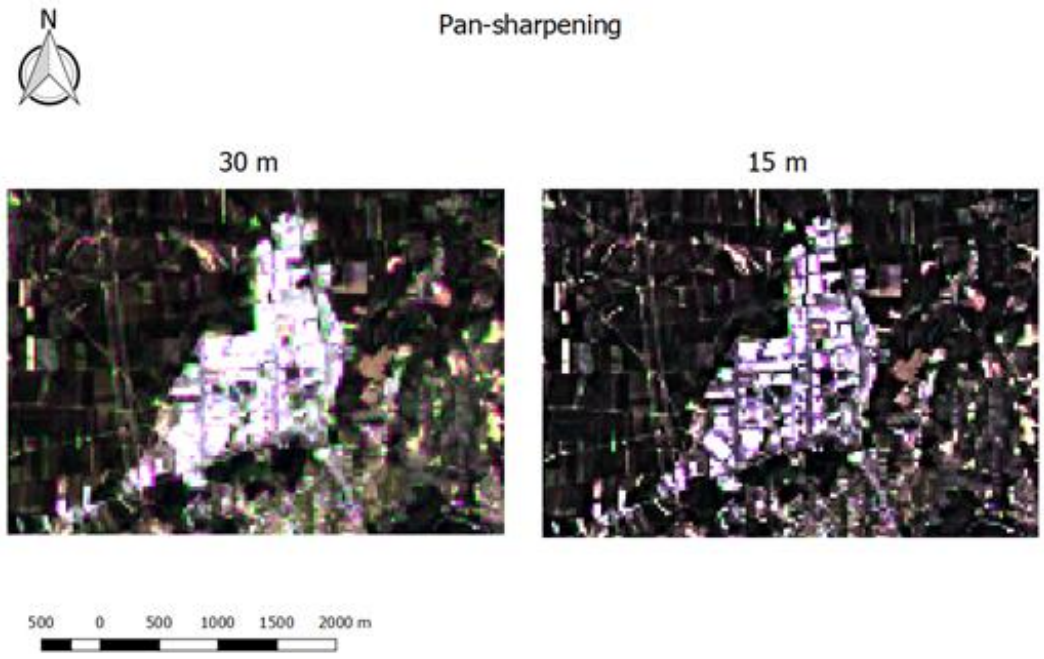


Figure 24 - Pan-sharpening example.

4.1.3 Classification results for Aveiro district

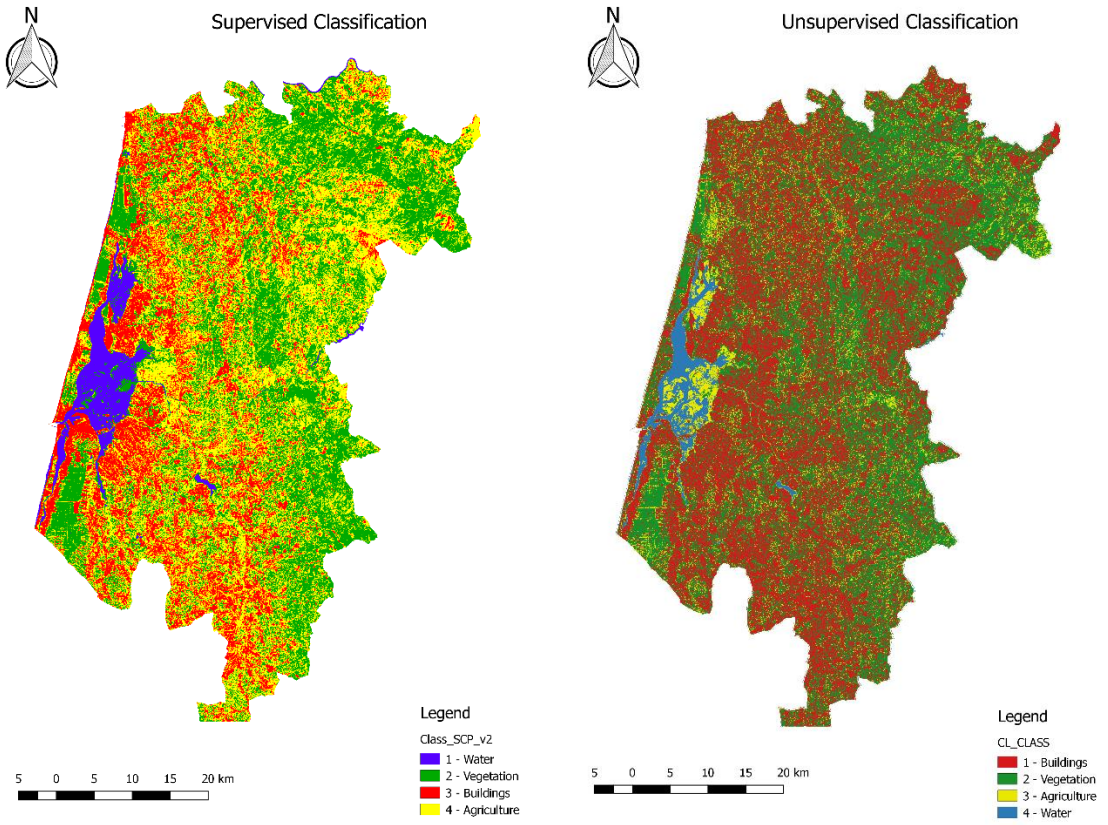
The unsupervised classification functionality was tested and the result was compared with a supervised classification obtained with the minimum distance algorithm from SCP tool [10].

Firstly, with PI2GIS processing group, a colour combination and pan-sharpening were applied to the bands 5, 4 and 3 in order to obtain a false colour combination. Afterwards, ROI's were selected using the Region Growing Algorithm. Those ROI's were used as a representation of the spectral characteristics of each class and gathered together in one shapefile (training area). The regions of interest considered a false colour combination are presented in the Figure 25.

Figure 25 - False colour combination and training area in black used in Supervised classification

Previously, a true colour combination (bands 4, 3 2) were used as an attempt for classification methods, but water and vegetation regions had similar spectral characteristics, so both classification methods classified incorrectly vegetation and water. This false colour combination is useful because vegetation regions are displayed in red. So, both classification methods can differentiate easily water from vegetation. The training areas were based on Corine Land Cover 2012 v.2 classes [44].

Four training areas were defined: water, vegetation, buildings and agriculture. Through the PI2GIS application, an unsupervised classification with the same number of classes was performed. Figure 26 presents the supervised and unsupervised classifications.



a **b**

Figure 26 - a) Supervised classification; b) Unsupervised classification.

By analysing both classifications, is possible to verify that vegetation and water areas are generally very similar in both classifications. The unsupervised classification showed a higher presence of buildings when compared with the supervised classification. On the other hand, agriculture areas are more presented in the supervised classification, when comparing with Corine Land Cover 2012 v.2 classes [44]. It is possible to conclude that the supervised classification makes a good recognition of the agriculture class. In the "Ria de Aveiro" the agriculture region was classified with the unsupervised classification method. While, in the same region salines are defined in Corine Land Cover 2012 v.2 [44].

Figure 27, shows the salines present in the corine land cover in grey (on the left) with the agriculture region classified in yellow (on the right).

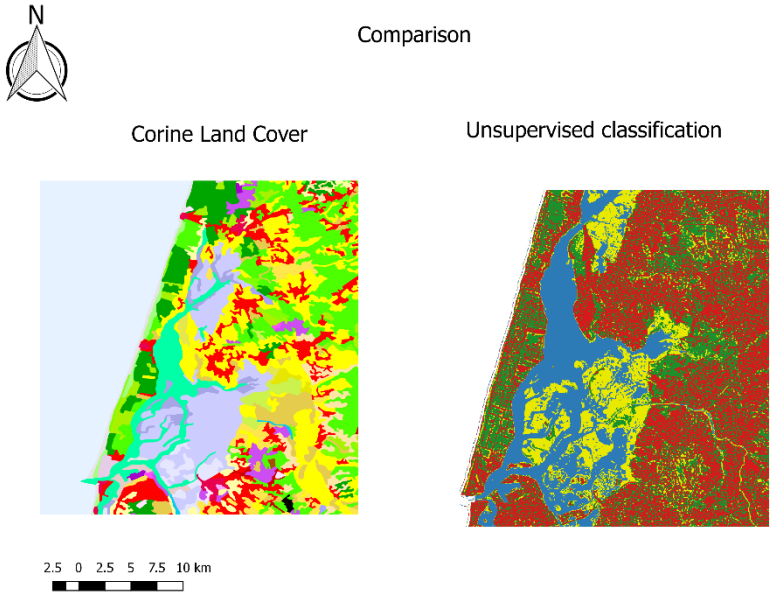


Figure 27 - On the left Corine Land Cover with salines in grey, on the right side agriculture region in yelow

Then, an accuracy assessment and a classification report (area in m^2 and percentage of land for each class) have been done for the supervised classification. On the unsupervised classification, there has been done only the classification report. Both tools, accuracy and classification report were performed using SCP plugin [10]. The accuracy assessment was important for supervised classification, because the training area used as the reference data can be compared with the maps generated with classification methods. The accuracy assessment generated from the supervised classification showed an overall accuracy of 94.5% and kappa statistic of 0.92, which indicates a good agreement between the maps generated from image and the reference data.

Afterwards, an error matrix was calculated for supervised classification (Table 2), which represents a class of comparison between classification and reference shapefile of training area (ROI's). The following error matrix represents the number of pixels classified correctly in the major diagonal.

Table 2 – Error matrix calculated for supervised classification with SCP for Aveiro

	Reference shapefile training area				
Raster	1 – Water	2 – Vegetation	3 - Buildings	4 - Agriculture	Total
1 - Water	9216	0	0	0	9216
2 - Vegetation	0	10651	0	0	10651
3 – Buildings	0	0	5167	0	5167
4 - Agriculture	0	1060	387	0	1447
Total	9216	11711	5554	0	26481

The classification report results for the supervised and unsupervised classification images are presented in the following tables.

Table 3 – Statistics obtained with SCP plugin for the supervised classification image

	Supervised Classification			
	Water	Vegetation	Buildings	Agriculture
Pixel sum	502287	4285725	2806668	5086104
Pixel percentage (%)	3.961 %	33.797 %	22.133 %	40.109 %
Area m^2	110887344.629	946137696.294	619613809.981	1122833294.640

Table 4 – Statistics obtained with SCP plugin for the unsupervised classification image obtained with PI2GIS

	Unsupervised classification			
	Buildings	Vegetation	Agriculture	Water
Pixel sum	1287061.000	1415742.000	268326.000	92616.000
Pixel percentage	41.062 %	45.168 %	8.651 %	2.955 %
Area m^2	1137028550.410	1250709231.360	237047290.546	81819770.955

4.2 Vila Nova de Gaia municipality

In Vila Nova de Gaia municipality, the same operations performed in Aveiro district were applied with very few differences. The Landsat 8 OLI image was the same.

4.2.1 Pre – Processing results for Gaia Municipality

In a similar way, pre-processing group operations were applied to the band 2, rescaling to 8-bits format and histograms. Figure 28 shows the histogram of band 2 created and saved as an image.

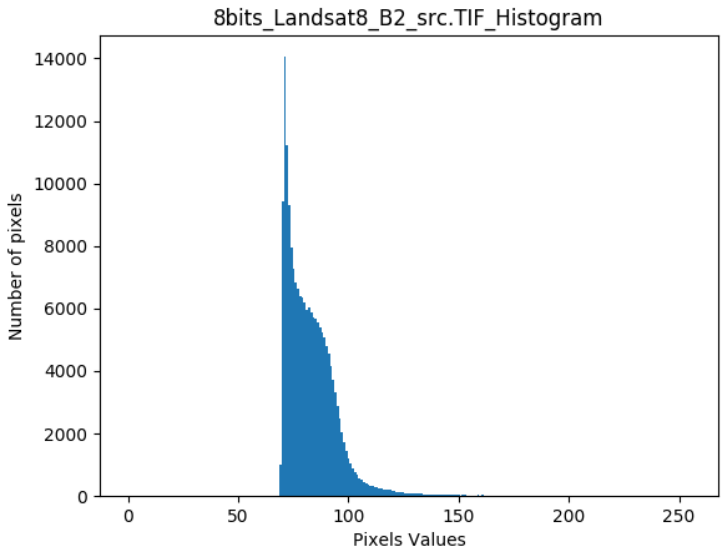
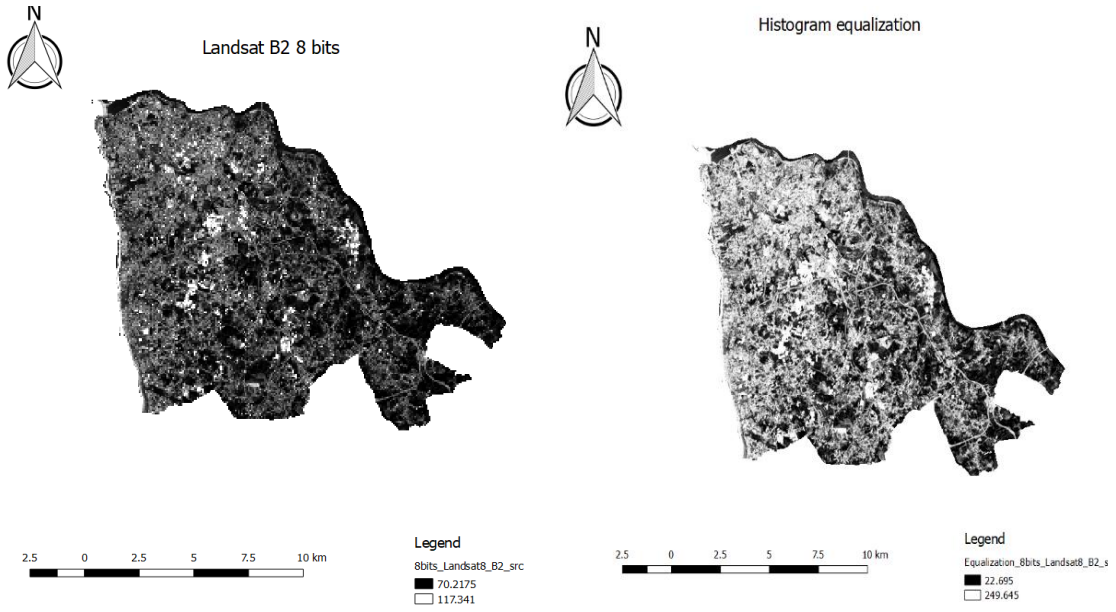


Figure 28 - Histogram for band 2

By analysing band 2 histogram is possible to verify that this image has a low brightness and low contrast. So, a possible solution for the contrast problem is the histogram equalization process. Figure 29 shows the difference between the original band 2 image and the same image resulted with more contrast.



a **b**

Figure 29 - a) Landsat B2 8 bits; b) Histogram equalization of Landsat B2 8 bits.

Furthermore, the following Figure 30 shows the histogram of image resulted by the equalization process.

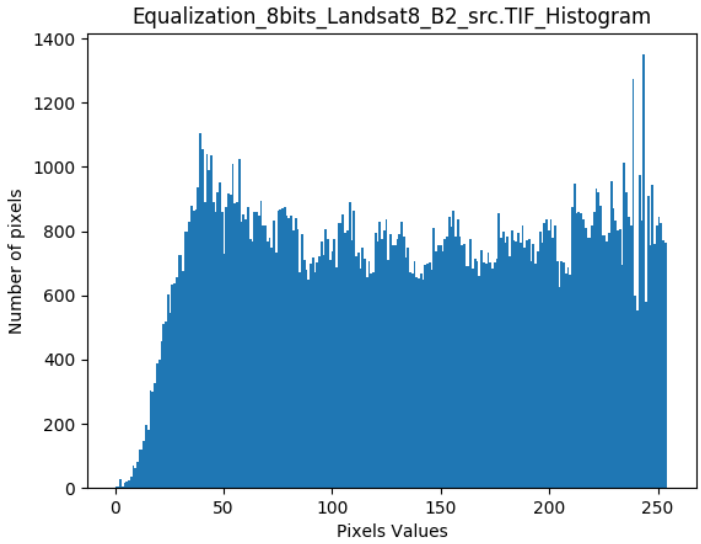
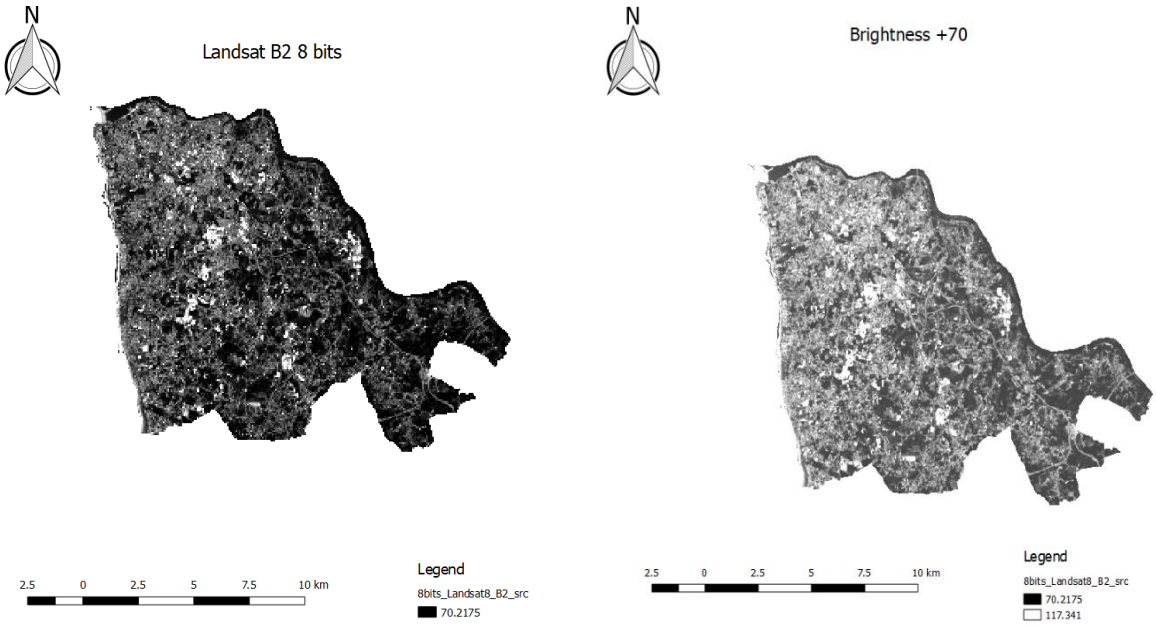


Figure 30 - Histograms resulted from the histogram equalization process

From the histogram present in Figure 30 is also possible to verify a contrast improvement resulted from histogram equalisation. This plugin can also increase the brightness by moving the handle to the left and clicking in the *Preview* button. Figure 31 shows the band 2 before and after the brightness was increased in 70 values of all pixels.



a **b**

Figure 31 - a) Landsat B2 8 bits; b) Landsat B2 8 bits with increased brightness

In the similar way with Aveiro, 3 different filters were applied, low pass, median and high pass filters. The Figures 32-34 show the original 8-bits image (on the left side) of a small area of Vila Nova de Gaia compared with the image resulted by the low pass filter with σ=1. As expected by analysing Figure 32 is possible to verify a smoothing effect which occurs after the application of the low pass filter.

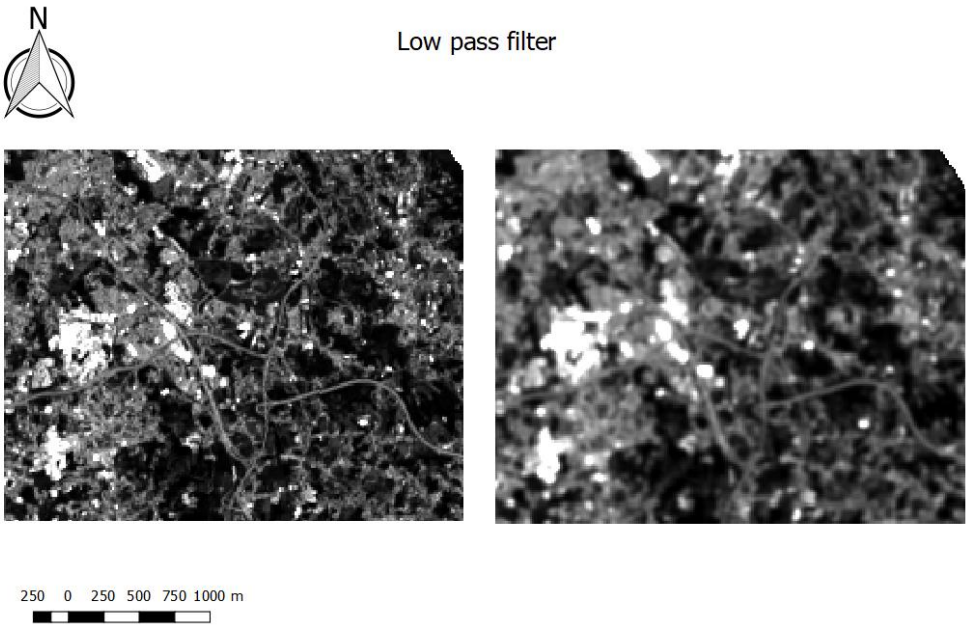


Figure 32 - Zoom in on the image before and after application Low pass filter

On the other hand, median filter with kernel size with size of 3×3 has the smoothing effect less present as it is shown in the Figure 33.

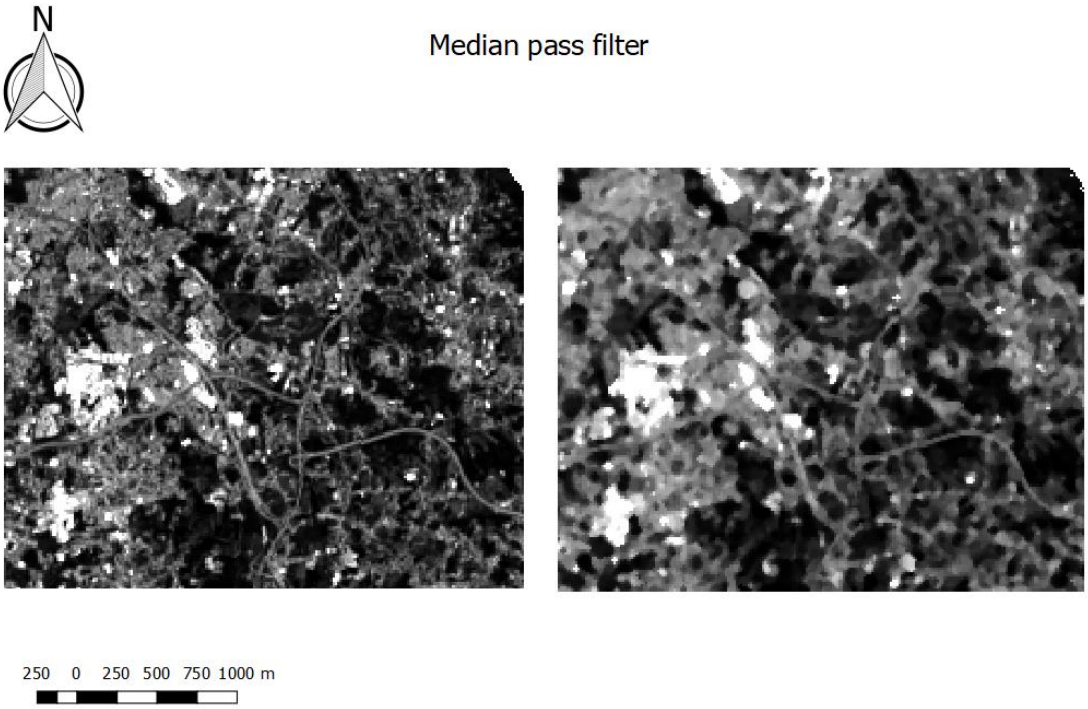


Figure 33 - Zoom in on the image before and after application Median filter

Finally, since the restricted area selected is high sloping variation region, high pass filter with σ= 3 is perfect for contour detection as it is shown in the Figure 34.

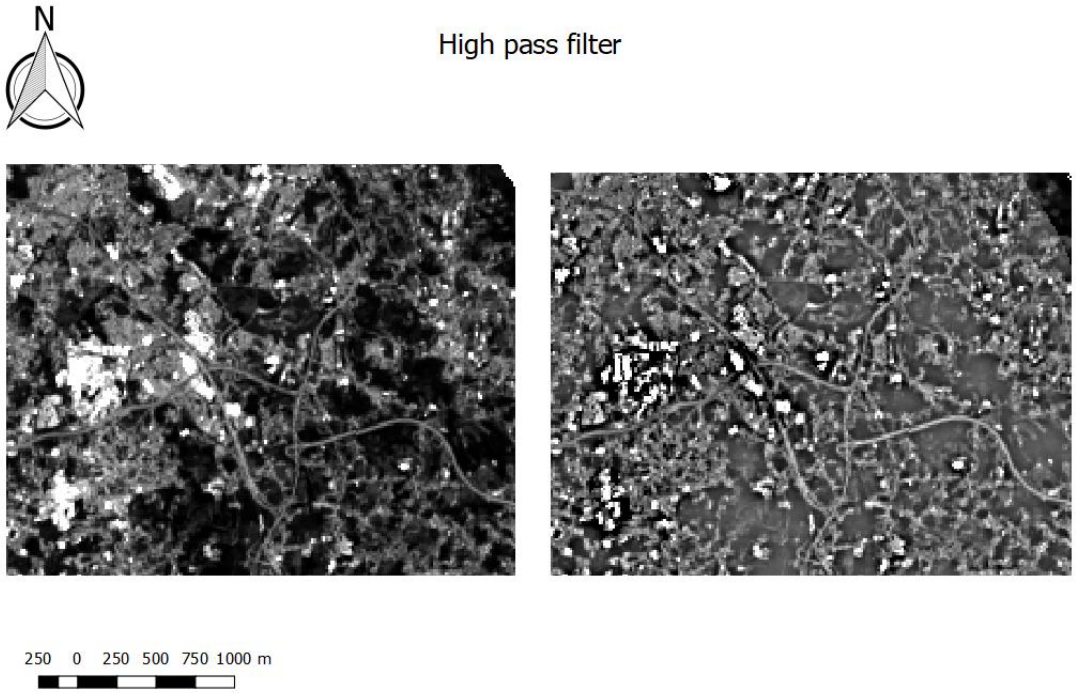


Figure 34 - Zoom in on the image before and after application High pass filter

4.2.2 Processing group for Vila Nova de Gaia municipality

In the processing group, RGB Composite, NDVI, EVI and NDWI indices were obtained. Figure 35 presents the results obtained.

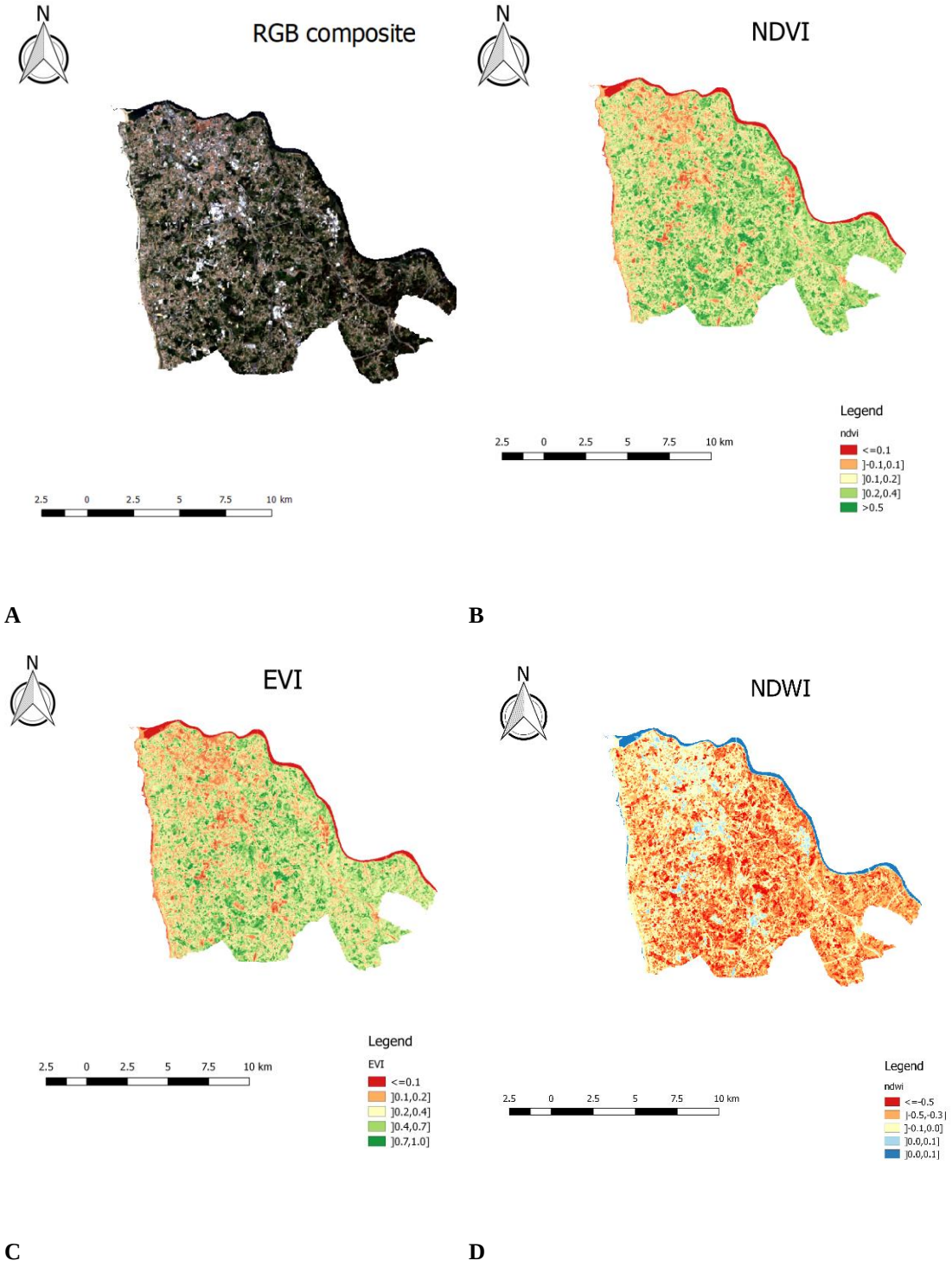


Figure 35 - a) RGB composite; b) NDVI map; c) EVI map; d) NDWI map

NDVI and EVI vegetation indices presented lower values in Vila Nova de Gaia when compared with Aveiro due to high urban pressure. Once NDWI is designed to maximize the reflectance of water, River Douro and several water bodies are easily

recognisable. After obtaining the indices, pan-sharpening functionality was tested. Figure 36 presents the difference.

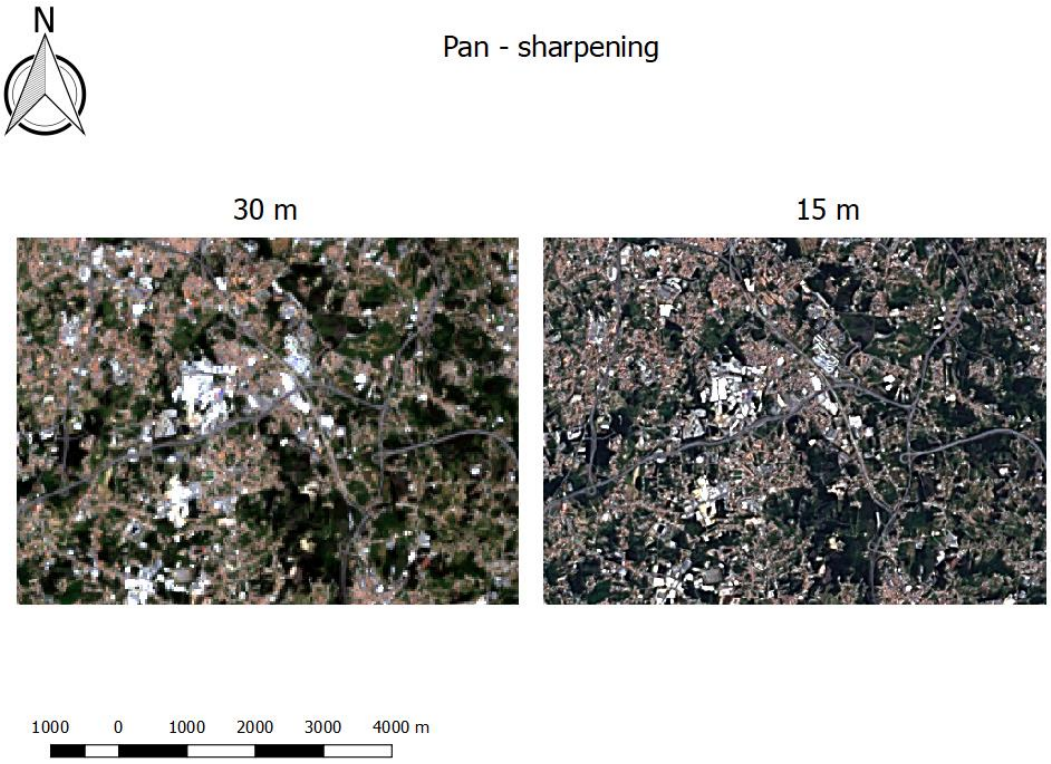


Figure 36 - Pan-sharpening example.

From Figure 42 is visually possible to spot the differences of detail between the original multi spectral lower resolution image (30 m) with the multi spectral high-resolution image (15 m) obtained by merging the original with high-resolution panchromatic band (band 8).

4.2.3 Classification results for Vila Nova de Gaia municipality

The unsupervised classification functionality was also tested for Vila Nova de Gaia municipality and the result was compared with a supervised classification obtained with the minimum distance algorithm from SCP tool. In a similar way of what was applied in Aveiro district, the same RGB combination (RGB=5,4,3) was used in the input image for both classification methods. Figure 37 shows the ROI's present in the training area obtained with SCP plugin and the false colour combination used for the supervised classification.

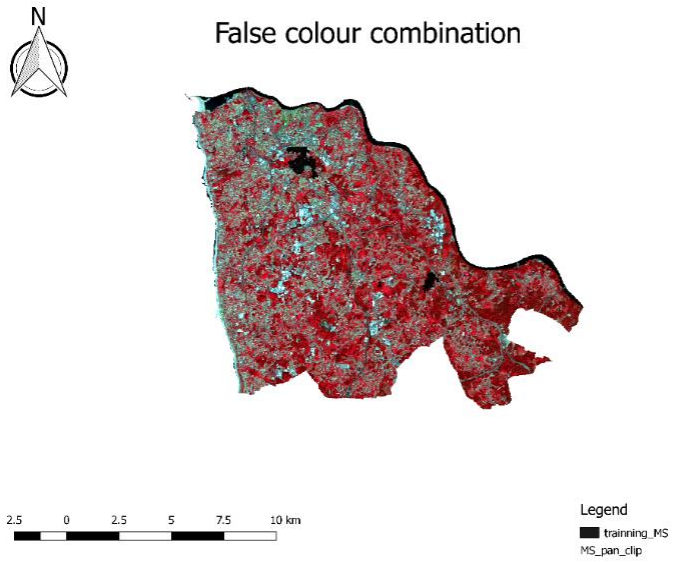
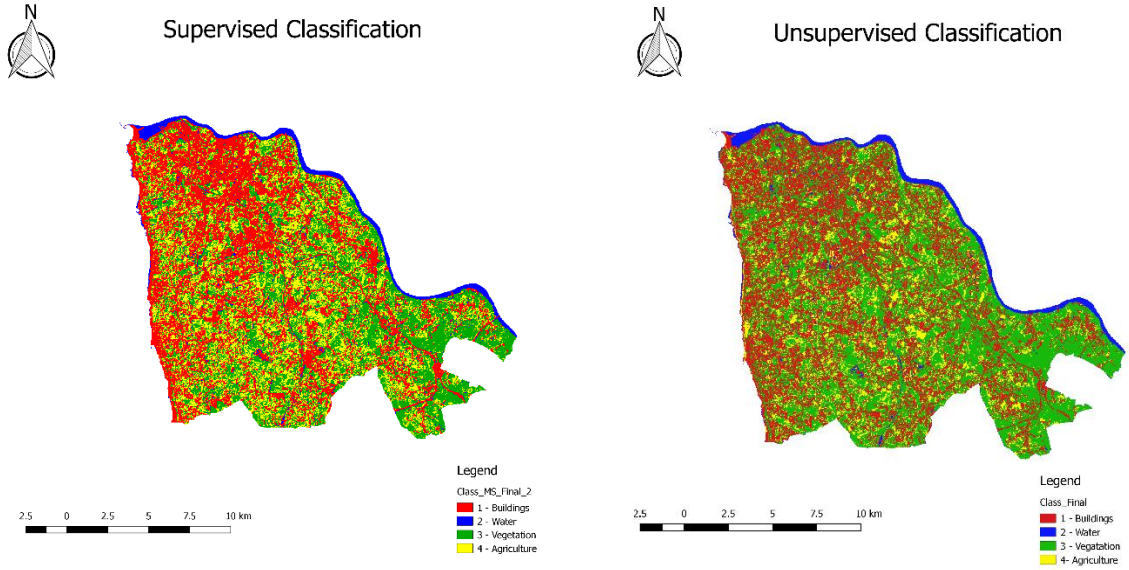


Figure 37 - False colour combination and training area in black used in Supervised classification

Four training areas were defined: water, vegetation, buildings and agriculture. Through the PI2GIS application, an unsupervised classification with the same number of classes was performed. Figures 38 shows the supervised and unsupervised classifications for Vila Nova de Gaia municipality.



a **b**

Figure 38 - a) Supervised classification b) Unsupervised classification

By analysing both Figures (Figure 38), supervised and unsupervised classification, it is possible to verify that the classes Vegetation Agriculture and Water are generally very

similar except from the buildings class that is denser in the supervised classification. It's possible to detect errors in both classifications in the coastal regions that are included in buildings class. This happens because sand and buildings have very similar spectral characteristics as shown in combination RGB.

An accuracy assessment and a classification report have been done same way as in the Aveiro district using tools from SCP plugin [10]. The accuracy assessment generated from the supervised classification for Vila Nova de Gaia municipality showed an overall accuracy of 98.5% and kappa statistic of 0.97, which indicates a good agreement between the maps generated from image and the reference data.

Afterwards, an error matrix was calculated, which represents a class of comparison between classification and reference shapefile of training area (ROI's) presented in the Table 5. The following error matrix represents the number of pixels classified correctly in the major diagonal.

Table 5 - Error matrix calculated for supervised classification with SCP for Vila Nova de Gaia

	Reference training area shapefile				
Raster	1	2	3	4	Total
1	2362	0	0	0	2362
2	0	162	0	0	162
3	52	0	623	0	675
4	0	0	0	204	204
Total	2414	162	623	204	3403

The classification report for the supervised and unsupervised classification images are presented in the Tables 6 and 7.

Table 6 – Statistics obtained with SCP plugin for the supervised classification image

	Supervised Classification			
	Buildings	Water	Vegetation	Agriculture
Pixel sum	275524	31888	250519	190381
Pixel percentage (%)	36.819 %	4.261 %	33.478 %	25.441 %
Area m^2	62012177.883	7177031.142	56384303.331	42849045.591

Table 7 - Statistics obtained with SCP plugin for the unsupervised classification image obtained with PI2GIS

	Unsupervised classification			
	Buildings	Water	Vegetation	Agriculture
Pixel sum	268423	26091	355158	97022
Pixel percentage	35.948 %	3.494 %	47.564 %	12.994 %
Area m^2	60441100.160	5874938.974	79971314.867	21846549.736

After checking the results of Table 6 and 7 is possible to verify that the classes buildings and water have very similar results which means that the unsupervised classification was accurate in recognising those elements that were clearly visible in the false colour combination as shown in Figure 36. On the other hand, vegetation and agriculture classes were slightly more difficult to be distinguished because both had very similar spectral characteristics, since false colour combination used was chose to enhance the high pics of vegetation reflectance. The Figure 39 shows how similar agriculture and vegetation regions were when performing the supervised classification.

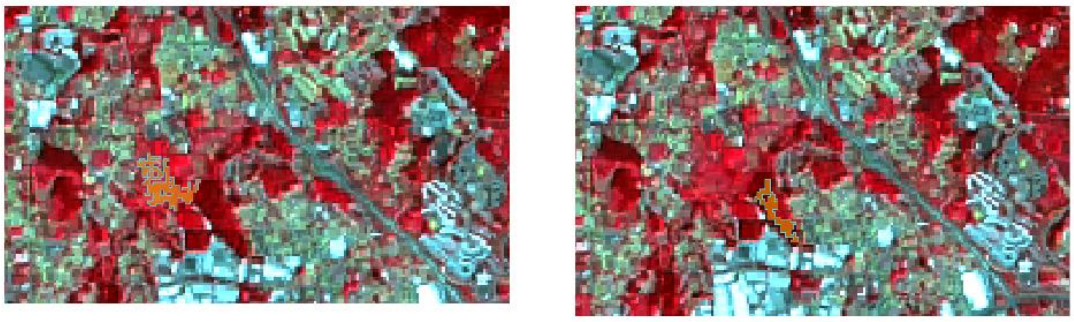


Figure 39 - Comparison of Training areas for Agriculture (on the left) and Vegetation (on the right) in the supervised classification

Since the training areas showed in Figure 46 had similar characteristics, it could explain why the k-means algorithm didn't have close results to the supervised classification.

5. Conclusions

The SCP plugin is an interesting open source remote sensing tool to perform supervised classification of remote sensing images providing several tools to pre-processing, processing, download, post-processing and raster calculation. Nevertheless, some tools are required in teaching classes of remote sensing and image processing, and there are not implemented in SCP, such as visualization of histograms, the application of filters, different image corrections, unsupervised classification and several environmental indices. This work has the main objective of include these functionalities in an open source application named PI2GIS. In this application, a toolbar was created with three buttons, pre-processing, processing and classification. The main advantage is the unsupervised classification implemented where only the number of classes are required to be predefined. The different functionalities implemented in the application were tested for two different regions from the North of Portugal such as Aveiro district, the one with bigger extension, and Vila Nova de Gaia municipality using the same Landsat 8 OLI images. In pre – processing and processing group the results obtained were very similar. The histogram equalisation, the filters and the conversion to reflectance with DOS1 correction were applied in both regions. In the processing group, different results were obtained for the vegetation index. Aveiro district has generally healthy vegetation because of high values of NDVI and EVI indices. A vast presence of water bodies such as the “Ria de Aveiro” reflected in the high values of NDWI. On Vila Nova de Gaia case, healthy vegetation is also visible, due to the high values of NDVI and EVI. Water bodies were detected particularly the river Douro with a high value of NDWI.

The same tools were applied to Aveiro and Vila Nova de Gaia but different results were obtained in the classification group. According to the statistics report, Vila Nova de Gaia municipality has higher similarity between the results obtained in the supervised and unsupervised classifications. A possible explanation would be that unsupervised classification has better results when it is tested with a smaller training set such as Vila Nova de Gaia municipality than with a district. The results obtained proved that the application developed could be a useful option to remote sensing teaching classes, regarding the tools implemented and the application availability given the fact it is free and open source. Therefore, it has the advantage to be adopted and/or improved with new tools/functionalities.

6. Future work

In the pre-processing group, the data processing such as the transformation to PTTM06 – ETRS89 (EPSG:3763) or any other coordinate system could be included. Further enhancement methods to improve contrast could also be included such as Optimal Linear Transformation or Gama Correction.

The implementation of a clipping process in the plugin process using a tool such as *gdalwarp* can be done in different sections of the process. The region of the shapefile of district or municipality can be clipped out of the original image. Due to that, it is possible to convert the undesirable values from the black outlier by selecting those values with the support of QDoubleSpinBox [30] and convert them into the value *nan.* The first section where the clipping process could be implemented in the plugin is in the beginning of the *convertionDN* tab, because the process of conversion to radiance or reflectance with DOS1 correction, requires the 16 – bits format for the input images. In this case, the n selected in QDoubleSpinBox would be 0 due to the black outlier.

After the application of the equations involved in the conversion to reflectance and radiance with DOS1 correction, two problems were verified: wrong coordinate systems of the output images and pixels with the value 0.01 around the district of municipality instead of Nan. Those problems could be solved by involving in the code the clipping process described before by selecting 0.01 with QDoubleSpinBox and the option *-t_srs* where the EPSG code would be selected in a different QSpinBox.

Furthermore, PI2GIS plugin can only rescale to reflectance with DOS1 correction to Landsat 8 (OLI) images, so in a future version could also have the possibility of applying this conversion with Sentinel -2A images.

In the processing group, the combination of bands with *Colour composite* results usually in a multi-spectral image from the black outlier around the region of interest with value 0 in black pixies, so the *Clip Raster by mask layer* algorithm from *GDAL library* can be implemented automatically in the code by applying the clipping described above with the value by replacing the value 0 to *nan.*

The calculation of other indices such as Soil Moisture Index (SMI) could also be implemented on PI2GIS [45].

The classification has been performed with the SCP plugin [10], the classification output is usually saved with the file in the format. *qml.* One possible improvement is

making the classification image generate *qml* file automatically. During the research for unsupervised classification methods, ISODATA was also considered as an alternative option. The ISODATA method is similar to k-means but more flexible. The classes with few elements are removed and the classes with centroids very close to each other are grouped together. In the future, the implementation would be useful and can be performed based on a function from *GitHub,* the *pyRadar's* documentation, which uses *GDAL* and *SCIPY* libraries and *Clustering* package [46].

Finally, an important tool that can be useful in the PI2GIS plugin is the one that creates a report classification with area and percentage of each class. In this work, the option available in the SCP plugin was used to compare results.

7. References

[1] Rahman, A., Kumar, S., Fazal, S., and Siddiqui, M.A., “Assessment of Land use/land cover Change in the North-West District of Delhi Using Remote Sensing and GIS Techniques,” J Indian Soc Remote Sens 40(4), 689–697 (2012).

[2] Kamel, M., and Ella, E., “Integration of Remote Sensing & GIS to Manage the Sustainable Development in the Nile Valley Desert Fringes of Assiut-Sohag Governorates, Upper Egypt,” J Indian Soc Remote Sens 44(5), 759–774 (2016).

[3] Yildiz, E., and Doker, M.F., “Monitoring urban growth by using segmentation-classification of multispectral Landsat images in Izmit, Turkey,” Environ Monit Assess 188, 393 (2016).

[4] García, P., and Pérez, E., “Mapping of soil sealing by vegetation indexes and built-up index: A case study in Madrid (Spain),” Geoderma 268, 100–107 (2016).

[5] Yan, L., He, R., Kašanin-Grubin, M., Luo, G., Peng, H., and Qiu, J., “The Dynamic Change of Vegetation Cover and Associated Driving Forces in Nanxiong Basin, China,” Sustainability 9, 443 (2017).

[6] Ringrose, S., Vanderpost, C., and Matheson, W., “Use of image processing and GIS techniques to determine the extent and possible causes of land management/fenceline induced degradation problems in the Okavango area, northern Botswana,” Journal of Remote Sensing, 18:11, 2337-2364 (2010).

[7] Usha, M., Anitha, K., and Iyappan, L., “Landuse Change Detection through Image Processing and Remote Sensing Approach: A Case Study of Palladam Taluk, Tamil Nadu,” International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622, Vol. 2, Issue 4, pp.289-294 (2012).

[8] Di Palma, F., Amato, F., Nolè, G., Martellozzo, F., and Murgante, B., “A SMAP Supervised Classification of Landsat Images for Urban Sprawl Evaluation,” ISPRS Int. J. Geo-Inf. 5(7), 109 (2016).

[9] Huth, J., Kuenzer, C., Wehrmann, T., Gebhardt, S., Tuan, V.Q., and Dech, S., “Land Cover and Land Use Classification with TWOPAC: towards Automated Processing for Pixel- and Object-Based Image Classification,” Remote Sensing 4(9), 2530-2553 (2012).

[10] Congedo, L., “Semi-Automatic Classification Plugin Documentation,” (2016).

< https://www.researchgate.net/profile/Luca_Congedo/publication/307593091_Semi-Automatic_Classification_Plugin_Documentation_Release_5361/links/57cb03bd08ae3ac722b1ec30/Semi-Automatic-Classification-Plugin-Documentation-Release-5361.pdf>

[11] USGS “Landsat Project Description”, <https://landsat.usgs.gov/landsat-project-description

[12] USGS “Landsat Missions Timeline”, < https://landsat.usgs.gov/landsat-missions-timeline>

[13] USGS “What are the band designations for the Landsat satellites ?” , https://landsat.usgs.gov/what-are-band-designations-landsat-satellites

[14] NLN for Remote Sensing, (1997)

< http://www.nln.geos.ed.ac.uk/courses/english/navig/beginf.htm>

[15] Gonzalez, R.C. and Woods, R.E., “Digital image processing,” Third Edition, Prentice Hall, (2008).

[16] Jain, A.K., “Fundamentals of Digital Image Processing Englewood Cliffs,” NJ: Prentice Hall, (1989).

[17] “Gaussian Filtering” – Lectures in The University of Auckland “

<https://www.cs.auckland.ac.nz/courses/compsci373s1c/PatricesLectures/Gaussian%20Filtering_1up.pdf>

[18] Solem, J E, “Programming Computer Vision with Python < https://www.safaribooksonline.com/library/view/programming-computer-vision/9781449341916/ch01.html>

[19] MacQueen, J.B., "Some Methods for classification and Analysis of Multivariate Observations,” Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley, University of California Press, 1:281-297 (1967).

[20] QGIS API, “QGIS API Documentation,” <http://www.qgis.org/api/> (22 May 2017). http://www.qgis.org/api/ (2017).

[21] PyQt4 API, “PyQt Class Reference,”

http://pyqt.sourceforge.net/Docs/PyQt4/classes.html.

(22 May 2017). http://pyqt.sourceforge.net/Docs/PyQt4/classes.html (2017).

[22] GDAL, “Geospatial Data Abstraction Library,” <http://www.gdal.org/> (22 May 2017). http://www.gdal.org/ (2017).

[23] Python, “Python Programming Language,” <http://python.org/> (2 May 2017). http://python.org/ (2017).

[24] Qt Designer, “Qt Documentation, Qt Designer Manual,” < http://doc.qt.io/qt-4.8/designer-manual.html> (10 May 2017). http://doc.qt.io/qt-4.8/designer-manual.html (2017).

[25] USGS “Earth Explorer” <https://earthexplorer.usgs.gov/>

[26] DGT, “Catálogo de Serviços de Dados Geográficos” (2017) <http://mapas.dgterritorio.pt/geoportal/catalogo.html>

[27] Matplotlib, “Matplotlib,” <http://matplotlib.org/> (3 May 2017), http://matplotlib.org/ (2017).

[28] Qslider class, “Qt Documentation, QSlider class,” <http://doc.qt.io/qt-5/qslider.html> (10 June 2017).

[29] QCheckBox Class, “Qt Documentation, QCheckBox Class,” <http://doc.qt.io/qt-5/qcheckbox.html#details > (28 August 2017).

[30] QSpinBox Class, “Qt Documentation”, < http://doc.qt.io/qt-4.8/qspinbox.html#details>

[31] J Gomez-Dans “Landsat DN to radiance script using GDAL and Numpy” GitHub < https://gist.github.com/jgomezdans/5488682>

[32] Landsat 8 Data Users Handbook, “USGS <https://landsat.usgs.gov/sites/default/files/documents/Landsat8DataUsersHandbook.pdf> (20 May 2017)

[33] Chavez, P.S., “Image-Based Atmospheric Corrections - Revisited and Improved Photogrammetric Engineering and Remote Sensing, [Falls Church, Va.],” American Society of Photogrammetry, 62, 1025- 1036 (1996).

[34] Center for Earth Observation, “Yale University” < http://yceo.yale.edu/how-convert-landsat-dns-top-atmosphere-toa-reflectance> (20 May 2017)

[35] Sobrino, J.; Jiménez-Muñoz, J. C. & Paolini, L. 2004. Land surface temperature retrieval from LANDSAT TM 5 Remote Sensing of Environment, Elsevier, 90, 434-440

[36] GRASS GIS 7.2.2svn Reference Manual, i.landsat.toar page

https://grass.osgeo.org/grass72/manuals/i.landsat.toar.html

[37] McFeeters, S.K., “The Use of Normalized Difference Water Index (NDWI) in the Delineation of Open Water Features,” International Journal of Remote Sensing, 17(7), 1425-1432 (1996).

[38] Solano, R., Didan, K., Jacobson, A., and Huete, A., “MODIS Vegetation Index User’s Guide (MOD13 Series),” Vegetation Index and Phenology Lab, University of Arizona, (2010).

[39] Didan, K., Munoz, A. B., Solano, R., and Huete, A., “MODIS vegetation index user’s guide (MOD13 Series),” <http://vip.arizona.edu/documents/MODIS/MODIS_VI_UsersGuide_June_2015_C6.pdf > (2015).

[40] OrfeoToolbox, “Superimpose sensor,” < https://www.orfeo-toolbox.org/Applications/Superimpose> (10 June 2017)

[41] OrfeoToolbox, “Pan-sharpening description,” <https://www.orfeo-toolbox.org/Applications/Pansharpening.html> (10 June 2017)

[42] OrfeoToolbox, “Unsupervised KMeans image classification description”,

< https://www.orfeo-toolbox.org/Applications/KMeansClassification.html>

[43] NASA, “NASA Earth observatory,” <https://earthobservatory.nasa.gov/Features/MeasuringVegetation/> (10 June 2017)

[44] DGT, “Direção Geral do Território. Cartografia de Uso e Ocupação do Solo,” (2017).

[45] Ivan Potić, Marko Bugarski, Jelena Matić-Varenica “Soil Moisture Determination Using Remote Sensing Data For The Property Protection and increase of Agriculture Production” < file:///C:/Users/rui_c/Downloads/Poster_Board_01-12-Matic_Varenica-305_paper%20(1).pdf>

[46] Pyradar, “Pyradar” http://pyradar-tools.readthedocs.io/en/latest/_modules/pyradar/classifiers/isodata.html

Annex

Script of the plugin PI2GIS developed in Python programming language of this thesis

```
# -*- coding: utf-8 -*-
"""
/***************************************************************************
 PI2GIS
                                 A QGIS plugin
 Processing Image to Geographical Information Systems – a learning
tool for QGIS
                             -------------------
        begin                : 2017-03-23
        git sha              : $Format:%H$
        copyright            : (C) 2017 by Rui Correia
        email                : rui_correia11@hotmail.com

 ***************************************************************************
 *****/

/***************************************************************************
 *
 *
 *   This program is free software; you can redistribute it and/or
modify  *
 *   it under the terms of the GNU General Public License as published
by  *
 *   the Free Software Foundation; either version 2 of the License, or
 *
 *   (at your option) any later version.
 *
 *
 *

 ***************************************************************************
 *****/
"""
from qgis.core import QgsRasterLayer, QgsApplication,
QgsBrightnessContrastFilter,QgsRasterPipe,QgsMapLayerRegistry

from PyQt4.QtCore import QSettings, QTranslator, qVersion,
QCoreApplication, QObject, SIGNAL,QDir, QSettings, QFileInfo
from PyQt4.QtGui import QColor, QFont, QMessageBox,
QTreeWidget,QAction, QIcon, QFileDialog

#from qgis.core import *

# Initialize Qt resources from file resources.py
import resources

# Import the code for the dialog
from PI2GIS_dialog import PI2GISDialog
```

```
from PI2GIS_dialog2 import PI2GISDialog2
from PI2GIS_dialog3 import PI2GISDialog3

from processing import *

import os, sys
import traceback
import optparse
import time

from osgeo import gdal, gdalnumeric, ogr, osr
import os.path
#import glob

#Math
import math

from sys import argv

#Importing Orfeo
import otbApplication

import subprocess
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import matplotlib.pylab as plt

from PIL import Image

#SciPY
from scipy import ndimage

#Bilioteca numpy
import numpy as np

try:
    from PyQt4.QtCore import QString
except ImportError:
    QString=str

class PI2GIS:
    """QGIS Plugin Implementation."""

    def __init__(self, iface):
        """Constructor.

        :param iface: An interface instance that will be passed to
this class
            which provides the hook by which you can manipulate the
QGIS
            application at run time.
        :type iface: QgsInterface
        """
        # Save reference to the QGIS interface
        self.iface = iface

        # initialize plugin directory
        self.plugin_dir = os.path.dirname(__file__)
```

```
        # initialize locale
        locale = QSettings().value('locale/userLocale')[0:2]
        locale_path = os.path.join(
            self.plugin_dir,
            'i18n',
            'PI2GIS_{}.qm'.format(locale))

        if os.path.exists(locale_path):
            self.translator = QTranslator()
            self.translator.load(locale_path)

            if qVersion() > '4.3.3':
                QCoreApplication.installTranslator(self.translator)

        # Declare instance attributes
        self.actions = []
        self.menu = self.tr(u'&PI2GIS')

        # TODO: We are going to let the user set this up in a future
iteration
        self.toolbar = self.iface.addToolBar(u'PI2GIS')
        self.toolbar.setObjectName(u'PI2GIS')


        # Create the dialog (after translation) and keep reference
        self.dlg = PI2GISDialog()
        self.dlg2 = PI2GISDialog2()
        self.dlg3 = PI2GISDialog3()

    # noinspection PyMethodMayBeStatic
    def tr(self, message):
        """Get the translation for a string using Qt translation API.

        We implement this ourselves since we do not inherit QObject.

        :param message: String for translation.
        :type message: str, QString

        :returns: Translated version of message.
        :rtype: QString
        """
        # noinspection PyTypeChecker,PyArgumentList,PyCallByClass
        return QCoreApplication.translate('PI2GIS', message)

    def add_action(
        self,
        icon_path,
        text,
        callback,
        enabled_flag=True,
        add_to_menu=True,
        add_to_toolbar=True,
        status_tip=None,
        whats_this=None,
        parent=None):
        """Add a toolbar icon to the toolbar.
```

```
        :param icon_path: Path to the icon for this action. Can be a
resource
            path (e.g. ':/plugins/foo/bar.png') or a normal file
system path.
        :type icon_path: str

        :param text: Text that should be shown in menu items for this
action.
        :type text: str

        :param callback: Function to be called when the action is
triggered.
        :type callback: function

        :param enabled_flag: A flag indicating if the action should be
enabled
            by default. Defaults to True.
        :type enabled_flag: bool

        :param add_to_menu: Flag indicating whether the action should
also
            be added to the menu. Defaults to True.
        :type add_to_menu: bool

        :param add_to_toolbar: Flag indicating whether the action
should also
            be added to the toolbar. Defaults to True.
        :type add_to_toolbar: bool

        :param status_tip: Optional text to show in a popup when mouse
pointer
            hovers over the action.
        :type status_tip: str

        :param parent: Parent widget for the new action. Defaults
None.
        :type parent: QWidget

        :param whats_this: Optional text to show in the status bar
when the
            mouse pointer hovers over the action.

        :returns: The action that was created. Note that the action is
also
            added to self.actions list.
        :rtype: QAction
        """

        icon = QIcon(icon_path)
        action = QAction(icon, text, parent)
        action.triggered.connect(callback)
        action.setEnabled(enabled_flag)

        if status_tip is not None:
            action.setStatusTip(status_tip)

        if whats_this is not None:
            action.setWhatsThis(whats_this)
```

```
        if add_to_toolbar:
            self.toolbar.addAction(action)

        if add_to_menu:
            self.iface.addPluginToMenu(
                self.menu,
                action)

        self.actions.append(action)

        return action

    def initGui(self):
        """Create the menu entries and toolbar icons inside the QGIS
GUI."""

        #C:\OSGeo4W64\apps\qgis\python\plugins\PI2GIS\icons

        icon_path = ':/plugins/PI2GIS/icons/icon.png'
        icon_path_2 = ':/plugins/PI2GIS/icons/icon2.png'
        icon_path_3 = ':/plugins/PI2GIS/icons/icon3.png'

        #----------------------------Activation of Push Buttons
"Run1"-------------------------------------------------
        #self.dlg.pushButton.clicked.connect(self.inputDir)#Caso tente
ler todas as imagens de uma vez
        self.dlg.pushButton.clicked.connect(self.inputDir)

        self.dlg.pushButton_2.clicked.connect(self.outputDir)
        self.dlg.pushButton_3.clicked.connect(self.inputshapefile)
        self.dlg.pushButton_4.clicked.connect(self.outputfiles_2)
        self.dlg.pushButton_5.clicked.connect(self.inputDir2)
        self.dlg.pushButton_6.clicked.connect(self.inputMTL)
        self.dlg.pushButton_9.clicked.connect(self.outputRefl)
        self.dlg.pushButton_10.clicked.connect(self.outrescale)
        self.dlg.pushButton_12.clicked.connect(self.outputPreview)

        # ----------------------------Activation of Push Buttons
"Run2"-------------------------------------------------

        self.dlg2.pushButton.clicked.connect(self.inputabc)
        self.dlg2.pushButton_2.clicked.connect(self.super)
        self.dlg2.pushButton_4.clicked.connect(self.Multi_input)
        self.dlg2.pushButton_5.clicked.connect(self.OUTPUT)
        self.dlg2.pushButton_6.clicked.connect(self.B8_Input)
        self.dlg2.pushButton_7.clicked.connect(self.Out_Pan)
        self.dlg2.pushButton_8.clicked.connect(self.inputshapefile8)


        #-----------------------------------------------Activation of
Push Buttons "Run 3"----------------------------

        #Unsupervised Classification

        self.dlg3.pushButton.clicked.connect(self.inshape3)
        self.dlg3.pushButton_2.clicked.connect(self.un_cl)
        self.dlg3.pushButton_3.clicked.connect(self.InputRGB)
```

```
        self.dlg3.pushButton_4.clicked.connect(self.Out_Un_Class)

        self.add_action(
            icon_path,
            text=self.tr(u''),
            callback=self.run,
            parent=self.iface.mainWindow())

        self.add_action(
            icon_path_2,
            text=self.tr(u''),
            callback=self.run2,
            parent=self.iface.mainWindow())

        self.add_action(
            icon_path_3,
            text=self.tr(u''),
            callback=self.run3,
            parent=self.iface.mainWindow())

        #--------------------------------------------------ComboBOX-----
-------------------------------------------------
        #--------------------------------------------------RUN1--------
-----------------------------------------------

        #Filters Methods-----------------------------------------------
--
        FM=[" ","Median","Low Pass","High Pass"]
        self.dlg.comboBox_2.addItems(FM)
        #DN converton
        convertion_type=["","Radiance","Reflectance"]
        self.dlg.comboBox_5.addItems(convertion_type)
        #Convertion From DN to Reflectance----------------------------
        Atmos=[" ","Radiance with DOS1","Reflectance with DOS1"]
        self.dlg.comboBox_3.addItems(Atmos)

        # ------------------------------------------------RUN2-------
-----------------------------------------------
        #Options Processing
        Options=[" ","Color Composite ","NDVI (A=B4,B=B5)","NDWI
(A=B5,B=B3)","EVI (A=B4,B=B5,C=B2)"]#
        # ,"SMI (A=NDVI Mtlfile=MTL)"]
        self.dlg2.comboBox.addItems(Options)

    def unload(self):
        """Removes the plugin menu item and icon from QGIS GUI."""
        for action in self.actions:
            self.iface.removePluginMenu(
                self.tr(u'&PI2GIS'),
                action)
            self.iface.removeToolBarIcon(action)
        # remove the toolbar
        del self.toolbar

    #--------------------------------------------------------------
Functions INITGUI RUN1 ----------------------------

    # ------------------------------------------Functions INITGUI
```

```
RUN1----------------------------------------------------------
    #Input first files
    def inputDir( self ):
        settings = QSettings()#Importar definições do PYQY
        #lastDir = settings.value( "/fTools/lastRasterDir", "." )
        inDir = QFileDialog.getExistingDirectory()#Consegui obter o
nome do diretório
        self.dlg.lineEdit.setText(inDir)#Escrito o nome do dir

        self.workDir = QDir( inDir )

        #Filtrar imagens
        self.workDir.setFilter( QDir.Files | QDir.NoSymLinks |
QDir.NoDotAndDotDot )
        nameFilter = [ "*.tif", "*.TIF", "*.hdf" ]
        fil = self.workDir.setNameFilters( nameFilter )

        self.inputFiles = self.workDir.entryList()#Lista Ligada

        rasters=self.inputFiles

        l = [" "] + ["All Bands"] + rasters

        self.dlg.comboBox_4.clear()

        self.dlg.comboBox_4.addItems(l)

        if len( self.inputFiles ) == 0:
            QMessageBox.warning( self.dlg, self.tr( "No raster files
found" ), self.tr( "There are no raster in this directory. Please
select another one." ) )
            self.inputFiles = None
            return


        return self.inputFiles#ficheiros filtrados da pasta numa lista
ligada


    def inputshapefile(self):
        inputFile = self.dlg.lineEdit_3.setText(
            QFileDialog.getOpenFileName(None, self.tr("Select the
input file"), '',
                                        self.tr("ESRI Shapefile
(*.shp)")))
        self.layer =
QgsVectorLayer(unicode(self.dlg.lineEdit_3.text()).encode('utf8'),
                                    self.dlg.lineEdit_3.text(), "ogr")

    # Output Histogram
    def outputDir(self):
        #self.dlg.lineEdit_2.setText(QFileDialog.getSaveFileName())
        settings = QSettings()#Importar definições do PYQY
        inDir = QFileDialog.getExistingDirectory()#Consegui obter o
nome do diretório
        self.dlg.lineEdit_2.setText(inDir)#Escrito o nome do dir
```

```
    def outputfiles_2(self):
        settings = QSettings()  # Importar definições do PYQY
        inDir = QFileDialog.getExistingDirectory()  # Consegui obter o
nome do diretório
        self.dlg.lineEdit_4.setText(inDir)  # Escrito o nome do dir


    def inputDir2(self):

        inDir = QFileDialog.getExistingDirectory()  # Consegui obter o
nome do diretório
        self.dlg.lineEdit_5.setText(inDir)  # Escrito o nome do dir

        self.workDir = QDir(inDir)

        # Filtrar imagens
        self.workDir.setFilter(QDir.Files | QDir.NoSymLinks |
QDir.NoDotAndDotDot)
        nameFilter = ["*.tif", "*.TIF", "*.hdf"]
        fil = self.workDir.setNameFilters(nameFilter)
        # selg.workDir esta filtrado e chama se fil
        self.inputFiles_2 = self.workDir.entryList()  # Lista Ligada

        if len(self.inputFiles_2) == 0:
            QMessageBox.warning(self.dlg, self.tr("No raster files
found"),
                                self.tr("There are no raster in this
directory. Please select another one."))
            self.inputFiles_2 = None
            return


        return self.inputFiles_2  # ficheiros filtrados da pasta numa
lista ligada

    def inputshapefile_2(self):
        inputFile = self.dlg.lineEdit_7.setText(
            QFileDialog.getOpenFileName(None, self.tr("Select the
input file"), '',
                                        self.tr("ESRI Shapefile
(*.shp)")))
        self.layer =
QgsVectorLayer(unicode(self.dlg.lineEdit_7.text()).encode('utf8'),
                                     self.dlg.lineEdit_7.text(), "ogr")

    def inputMTL(self):
        self.dlg.lineEdit_6.setText(QFileDialog.getOpenFileName())

    def outputRefl(self):
        settings = QSettings()  # Importar definições do PYQY
        inDir = QFileDialog.getExistingDirectory()  # Consegui obter o
nome do diretório
        self.dlg.lineEdit_9.setText(inDir)  # Escrito o nome do dir

    def OUT_EPSG(self):
        settings = QSettings()  # Importar definições do PYQY
        inDir = QFileDialog.getExistingDirectory()  # Consegui obter o
nome do diretório
```

```
        self.dlg.lineEdit_8.setText(inDir)  # Escrito o nome do dir

    def outrescale(self):
        settings = QSettings()  # Importar definições do PYQY
        inDir = QFileDialog.getExistingDirectory()  # Consegui obter o
nome do diretório
        self.dlg.lineEdit_10.setText(inDir)  # Escrito o nome do dir

    #-----------------------------------------------Functions INITGUI
RUN2-------------------------------------------------------

    def inputabc(self):
        settings = QSettings()  # Importar definições do PYQY
        # lastDir = settings.value( "/fTools/lastRasterDir", "." )
        inDir = QFileDialog.getExistingDirectory()  # Consegui obter o
nome do diretório
        self.dlg2.lineEdit.setText(inDir)  # Escrito o nome do dir

        self.workDir = QDir(inDir)

        # Filtrar imagens
        self.workDir.setFilter(QDir.Files | QDir.NoSymLinks |
QDir.NoDotAndDotDot)
        nameFilter = ["*.tif", "*.TIF", "*.hdf"]
        fil = self.workDir.setNameFilters(nameFilter)

        self.inputFiles = self.workDir.entryList()  # Lista Ligada

        RASTERS = self.inputFiles
        l = [" "] + RASTERS

        self.dlg2.comboBox_3.clear()
        self.dlg2.comboBox_4.clear()
        self.dlg2.comboBox_5.clear()

        self.dlg2.comboBox_3.addItems(l)
        self.dlg2.comboBox_4.addItems(l)
        self.dlg2.comboBox_5.addItems(l)

        if len(self.inputFiles) == 0:
            QMessageBox.warning(self.dlg, self.tr("No raster files
found"),
                                self.tr("There are no raster in this
directory. Please select another one."))
            self.inputFiles = None
            return


        return self.inputFiles  # ficheiros filtrados da pasta numa
lista ligada

    def inputshapefile8(self):
        inputFile = self.dlg2.lineEdit_8.setText(
            QFileDialog.getOpenFileName(None, self.tr("Select the
input file"), '',
                                        self.tr("ESRI Shapefile
(*.shp)")))
        self.layer =
```

```
QgsVectorLayer(unicode(self.dlg2.lineEdit_8.text()).encode('utf8'),
                                    self.dlg2.lineEdit_8.text(),
"ogr")

    def OUTPUT(self):
        self.dlg2.lineEdit_5.setText(QFileDialog.getSaveFileName())


    def Multi_input(self):
        self.dlg2.lineEdit_4.setText(QFileDialog.getOpenFileName())

    def B8_Input(self):
        self.dlg2.lineEdit_6.setText(QFileDialog.getOpenFileName())

    def super(self):
        self.dlg2.lineEdit_2.setText(QFileDialog.getSaveFileName())

    def Out_Pan(self):
        self.dlg2.lineEdit_7.setText(QFileDialog.getSaveFileName())


    #-----------------------------------------------Functions INITGUI
RUN3-----------------------------------------

    def inshape3(self):
        inputFile = self.dlg3.lineEdit.setText(
            QFileDialog.getOpenFileName(None, self.tr("Select the
input file"), '',
                                        self.tr("ESRI Shapefile
(*.shp)")))
        self.layer =
QgsVectorLayer(unicode(self.dlg3.lineEdit.text()).encode('utf8'),
                                    self.dlg3.lineEdit.text(), "ogr")

    def un_cl(self):
        self.dlg3.lineEdit_2.setText(QFileDialog.getSaveFileName())

    def InputRGB(self):
        self.dlg3.lineEdit_3.setText(QFileDialog.getOpenFileName())

    def Out_Un_Class(self):
        self.dlg3.lineEdit_4.setText(QFileDialog.getSaveFileName())

    #-------------------------------------------------RUN1-------------
-------------------------------------------------
    def run(self):
        self.dlg.show()
        QObject.connect(self.dlg.button_box, SIGNAL("accepted()"),
self.preprocessing)
        QObject.connect(self.dlg.button_box, SIGNAL("rejected()"),
self.exit)
        QObject.connect(self.dlg.button_box_2, SIGNAL("accepted()"),
self.convertionDN)
        QObject.connect(self.dlg.button_box_2, SIGNAL("rejected()"),
self.exit)

    def exit(self):
        # Close the widnow
```

```
        self.dlg.close()


    def preprocessing(self):#Pre Processing Image

        inputLayer = self.dlg.lineEdit.text()#Name until directory
thanks to inDir

        out_tx_hist = self.dlg.lineEdit_2.text()  # In this case it is
directory where I want to put histogram images
        # Obtain with outputDir only strings

        out_rescale = self.dlg.lineEdit_10.text()

        inshape=self.dlg.lineEdit_3.text()

        rasters = self.inputFiles

        for i in range(len(rasters)):

            #------------------------ 1. Rescale all bands -----------
-------------------------------
            if not(len(out_rescale))==0:

                image_in =os.path.join(str(inputLayer),
str(rasters[i])) #"path_input_image"

out=QFileInfo(QgsApplication.qgisUserDbFilePath()).path()+"/out"

                subprocess.call(["gdal_translate.exe","-of","GTiff","-
ot","Float32","-scale","-co", "TFW=YES",image_in,out])

                subprocess.call(["gdalwarp.exe","-srcnodata", "255",
"-dstnodata", "nan","-crop_to_cutline","-cutline",inshape,out,

os.path.join(str(out_rescale),"8bits_"+str(rasters[i]))])

            # ---------------------1. Histograms of all bands --------
--------------------------------

            if not(len(out_tx_hist))==0:


                im =
gdal.Open(os.path.join(str(inputLayer),str(rasters[i])))
                im = np.array(im.GetRasterBand(1).ReadAsArray())
                im = im.flatten()  # copy of the array collapsed into
one dimension.

                plt.Figure()#show image
                plt.title(str(rasters[i])+"_Histogram")
                plt.ylabel("Number of pixels")
                plt.xlabel("Pixels Values")
                #,bins=256 I culd use bins=None
                plt.hist(im,bins=256,range=[0,255])#,255])
                plt.show()

                s = str(rasters[i])
```

```
                H="Hist_"
                plt.savefig(os.path.join(str(out_tx_hist),H+s)) #
Guardar de ficheiro de texto

            # ----------------  3, 4 and 5 Pre - Processing Operations
-----------------------------------
            def pre_operations(index):
                # QMessageBox.about(self.dlg, 'rasters',
str(len(rasters)))
                # QMessageBox.about(self.dlg, 'selected_raster_index',
str(selected_raster_index))

                im = gdal.Open(os.path.join(str(inputLayer),
str(rasters[index])))
                # QMessageBox.about(self.dlg, 'inputband',
str(rasters[selected_raster_index]))
                x = im.RasterXSize
                y = im.RasterYSize
                data = np.array(im.GetRasterBand(1).ReadAsArray())
                geo = im.GetGeoTransform()
                srs = im.GetProjectionRef()  # Projection
                # Processing.initialize()

                # output
                Pre_Processed_tx = self.dlg.lineEdit_4.text()

                # Index 2 - Histogram equalization ------------------
----------------
                # Task: Trasnlate to matplotlib qgis capacities
                def histeq(data):#, nbr_bins=255):

                    #Safari ----------------------------------------
---
                    """  Histogram equalization of a grayscale image.
"""

                    # get image histogram
                    # usage of numpy library
                    nbr_bins = 256

                    #QMessageBox.about(self.dlg, 'shape',
str(data.shape))

                    imhist,bins
=np.histogram(data.flatten(),256,[0,256])
                    #QMessageBox.about(self.dlg, 'bins', str(bins))

                    cdf = imhist.cumsum()# cumulative distribution
function
                    cdf = 255 * cdf / cdf[-1]#normalize

                    data2 = np.interp(data.flatten(), bins[:-1], cdf)

                    return data2.reshape(data.shape)

                if  self.dlg.checkBox.isChecked():
                    data = histeq(data)  # Call the function
                    H = "Equalization_"
```

```
                else:
                    pass

                # ----------------------FILTERS--------------------
---------------
                # Call FM function according to selection in ComboBox-
---------------------
                selectedFMindex = self.dlg.comboBox_2.currentIndex()
                sigma = self.dlg.spinBox_2.value()
                if selectedFMindex == 0:
                    pass  # No FM function selected
                if selectedFMindex == 1:# Median
                    data = ndimage.filters.median_filter(data,
size=(sigma, sigma))
                    H="Median_"
                if selectedFMindex == 2:# Low Pass
                    H="Low Pass_"
                    data = ndimage.gaussian_filter(data, sigma)
                if selectedFMindex == 3:# High Pass countor detection
                    H = "High Pass_"
                    lowpass = ndimage.gaussian_filter(data, sigma)
                    data = data - lowpass
                # ---------------------Show and Save of Pre -
Processed Image ----------------
                Pre_Processed_tx = self.dlg.lineEdit_4.text()

                driver = gdal.GetDriverByName("GTiff")
                outData =
driver.Create(os.path.join(str(Pre_Processed_tx),H +
str(rasters[index])), x, y, 1,
                                          gdal.GDT_Float32)
                outData.SetProjection(srs)
                outData.SetGeoTransform(geo)
                outData.GetRasterBand(1).WriteArray(data)
                outData = None
                # ----------------------------------------------------
---------------------------------------------------
            selected_raster_index = self.dlg.comboBox_4.currentIndex()

        if selected_raster_index == 1:
            # Read all bands
            for i in range(len(rasters)):
                pre_operations(i)  # Function referes to rasters list
                # QMessageBox.about(self.dlg,str(i),str(rasters(i)))
        elif selected_raster_index > 1:
            pre_operations(selected_raster_index - 2)
            # Because this selected_raster_index refers to the list 1
        elif selected_raster_index==0:
            pass

        QMessageBox.about(self.dlg, 'pre operations', 'Finished')

    def outputPreview(self):

        # Temporary directory
        light_preview =
os.path.join(QFileInfo(QgsApplication.qgisUserDbFilePath()).path(),
'light_preview')
```

```
        inputLayer = self.dlg.lineEdit.text()  # Name until directory
thanks to inDir

        rasters = self.inputFiles
        l = [" "]+ ["All Bands"] + rasters

        selected_raster_index = self.dlg.comboBox_4.currentIndex()

        def light(i):

            # 3. Light
Correction.....................................................................
.............
            light_value = self.dlg.horizontalSlider.value()
            #if light_value != 0:
                #data = data + light_value

            file_Info = QFileInfo(str(os.path.join(str(inputLayer),
str(rasters[i]))))
            layer_name = file_Info.baseName()
            layer = QgsRasterLayer(str(os.path.join(str(inputLayer),
str(rasters[i]))), layer_name)

            # define the filter
            brightnessFilter = QgsBrightnessContrastFilter()
            brightnessFilter.setBrightness(light_value)

            #QMessageBox.about(self.dlg, 'brightFilter ',
str(brightnessFilter))

            pipe = QgsRasterPipe()
            #QMessageBox.about(self.dlg, 'pipe', str(pipe))

            # assign filter to raster pipe
            layer.pipe().set(brightnessFilter)
            # QMessageBox.about(self.dlg, 'Linha 833', str(layer))

            # apply changes
            layer.triggerRepaint()
            #QMessageBox.about(self.dlg, 'layer', str(layer))
            QgsMapLayerRegistry.instance().addMapLayer(layer)

        if selected_raster_index == 1:
            # Read all bands
            for i in range(len(rasters)):
                light(i)  # Function referes to rasters list
        elif selected_raster_index > 1:
            light(selected_raster_index - 2)
            # Because this selected_raster_index refers to the list l
        elif selected_raster_index == 0:
            pass

    ####Convertion DN

    def convertionDN(self):
        #----------LANDSAT 8 (OLI)----------------
```

```
        #Source

        #Convertion to radiance with git hub

        #https://gist.github.com/jgomezdans/5488682

        #Author:  J Gomez-Dans <j.gomez-dans@ucl.ac.uk>

        #Directory of LandBands in DN

        #From UI file

        InputLayer2 = self.dlg.lineEdit_5.text()

        # Read MTL file

        MTL = self.dlg.lineEdit_6.text()

        inshape2=self.dlg.lineEdit_7.text()

        Out_Refle = self.dlg.lineEdit_9.text()

        out_epsg=self.dlg.lineEdit_8.text()


        if not (len(InputLayer2) and len(MTL)) == 0:

            fp = open(str(MTL), 'r')

            #Creation of dictionaries for Reflectance multi and add
            multi = {}
            add = {}
            se = {}
            d = {}

            multi_rad = {}
            add_rad = {}

            rad_max = {}
            refle_max = {}
            #rad_min= {}


            for line in fp:

                if (line.find("REFLECTANCE_MULT_BAND") >= 0):
                    s = line.split("=")  # Split by equal sign
                    the_band = int(s[0].split("_")[3])  # Band number
as integer
                    if the_band in [1,2, 3, 4, 5, 6, 7,8,9,10,11,12]:
# Is this one of the bands we want?
                        multi[the_band] = float(s[-1])  # Attribui um
valor a uma chave como float

                if (line.find("REFLECTANCE_ADD_BAND") >= 0):
                    s = line.split("=")  # Split by equal sign
                    the_band = int(s[0].split("_")[3])  # Band number
as integer
```

```
                    if the_band in [1,2, 3, 4, 5, 6, 7,8,9,10,11,12]:
# Is this one of the bands we want?
                        add[the_band] = float(s[-1])  # Get constant
as float

                if (line.find("REFLECTANCE_MAXIMUM_BAND")>=0):
                    s = line.split("=")  # Split by equal sign
                    the_band = int(s[0].split("_")[3])  # Band number
as integer
                    if the_band in [1,2, 3, 4, 5, 6, 7,8,9,10,11,12]:
# Is this one of the bands we want?
                        refle_max[the_band] = float(s[-1])


                if (line.find("RADIANCE_MULT_BAND") >= 0):
                    s = line.split("=")  # Split by equal sign
                    the_band = int(s[0].split("_")[3])  # Band number
as integer
                    if the_band in [1,2, 3, 4, 5, 6, 7,8,9,10,11,12]:
# Is this one of the bands we want?
                        multi_rad[the_band] = float(s[-1])

                if (line.find("RADIANCE_ADD_BAND") >= 0):
                    s = line.split("=")  # Split by equal sign
                    the_band = int(s[0].split("_")[3])  # Band number
as integer
                    if the_band in [1,2, 3, 4, 5, 6, 7,8,9,10,11,12]:
# Is this one of the bands we want?
                        add_rad[the_band] = float(s[-1])


                if (line.find("RADIANCE_MAXIMUM_BAND")>=0):
                    s = line.split("=")  # Split by equal sign
                    the_band = int(s[0].split("_")[3])  # Band number
as integer
                    if the_band in [1,2, 3, 4, 5, 6, 7,8,9,10,11,12]:
# Is this one of the bands we want?
                        rad_max[the_band] = float(s[-1])


                if (line.find("SUN_ELEVATION")>= 0):
                    s = line.split("=")  # Split by equal sign
                    #QMessageBox.about(self.dlg, 's', str(s))
                    se=float(s[1])
                    #QMessageBox.about(self.dlg, 'se', str(se))
                    # Solar zenith angle θs  = 90° - θe
                    ss = 90 - se

                if (line.find("EARTH_SUN_DISTANCE")>=0):
                    s = line.split("=")  # Split by equal sign
                    d=float(s[-1])


        RastersDN = self.inputFiles_2

        if not (len(InputLayer2)) == 0:
            #Has to have landbands in
            for i in range(len(RastersDN)):
```

```
                #Temporary
                band=i+2#Works for 2,3,4,5,6,7 and 8


image_in=os.path.join(str(InputLayer2),str(RastersDN[i]))

                s = str(RastersDN[i])

                g=gdal.Open(image_in)
                #QMessageBox.about(self.dlg, 'Raster',
str(RastersDN[i]))

                ## find DNmin in raster for DOS1
                x_size, y_size = g.RasterXSize, g.RasterYSize

                data = np.array(g.GetRasterBand(1).ReadAsArray())  #
,dtype=float)#arriscar o float

                DNmin=np.amin(data)


                geo = g.GetGeoTransform()

                #-----------DOS 1 Landsat_8_OLI------------

                #This was done for the band 2 until 8


                selectedCTindex = self.dlg.comboBox_5.currentIndex()

                if selectedCTindex==0:
                    pass
                if selectedCTindex==1:


im=self.convertionRad(data,multi_rad[band],add_rad[band])

                    H = "Rad_"

                if selectedCTindex==2:

im=self.convertionRefle(data,multi[band],add[band],se)
                    im[im>1]=np.nan
                    H = "Rfl_"


                selectedDOSindex=self.dlg.comboBox_3.currentIndex()

                if selectedDOSindex==0:
                    pass
                if selectedDOSindex==1:#Radiance


Ldos=(0.01*self.ESUN(d,refle_max[band],rad_max[band])*(math.cos(math.r
adians(ss))))/math.pi*d**2
                    Lmin=multi_rad[band]*DNmin+add_rad[band]
                    im=Lmin-Ldos
```

```
                if selectedDOSindex==2:
                    E=self.ESUN(d, refle_max[band], rad_max[band])
                    Lp = multi_rad[band] * DNmin + add_rad[band]-0.01
* E * (math.cos(math.radians(ss))) / (math.pi * math.pow(d,2))

Ll=self.convertionRad(data,multi_rad[band],add_rad[band])
                    im=(math.pi*(Ll-
Lp)*math.pow(d,2))/(E*math.cos(math.radians(ss)))
                    im[im > 1] = np.nan


                    H="DOS1_"

                # QMessageBox.about(self.dlg, 'teste', str(im))
                gdal.AllRegister()

                # Output GDAL way
                # Create an output imagedriver
                driver = gdal.GetDriverByName("GTiff")

                outDN = os.path.join(str(Out_Refle), H +
str(RastersDN[i]))

                #s is the name of the file like Landsat_B*
                #outData = driver.Create(os.path.join(str(Out_Refle),
H + s), x_size, y_size, 1,gdal.GDT_Float32)
                outData = driver.Create(outDN,x_size, y_size, 1,
gdal.GDT_Float32)

                outData.GetRasterBand(1).WriteArray(im)

                outData.SetGeoTransform(geo)

                outData = None

                QMessageBox.about(self.dlg, 'Convertion ', "complete")

    def convertionRad(self,dat, mr, ar):
        radi = mr * dat + ar
        return radi

    # Application of formula of convertion

    def convertionRefle(self,dat, mp, ap, solar):
        # img is an array
        # Band=np.array(img,dtype=float)
        refle = mp * dat + ap
        # Sun elevation
        # x=59.18937092
        Refle = refle / (math.sin(math.radians(solar)))
        return Refle  # .reshape(img.shape)

        # DOS1 ----------------------------------------------------
------------

    # (π*d2)*RADIANCE_MAXIMUM/REFLECTANCE_MAXIMUM
    def ESUN(self,dis, pmax, rmax):  # ban is the index of band
        # pmax is REFLECTANCE_MAXIMUM_BAND
```

```
        # rmax is RADIANCE_MAXIMUM_BAND
        return (math.pi * math.pow(dis,2)) * (rmax / pmax)



    # ----------------------------------------------RUN2------------
-----------------------------------------------
    def run2(self):
        self.dlg2.show()

        QObject.connect(self.dlg2.buttonBox, SIGNAL("accepted()"),
self.processing)


    def processing(self):
        # Run the dialog event loop

        gdal.AllRegister()



        INPUT = self.dlg2.lineEdit.text()


        if not(len(INPUT)==0):

            inshape8=self.dlg2.lineEdit_8.text()


            RASTERS = self.inputFiles


            selectedAindex = self.dlg2.comboBox_3.currentIndex()
            selectedBindex = self.dlg2.comboBox_4.currentIndex()
            selectedCindex = self.dlg2.comboBox_5.currentIndex()

            #A= Image.open(str(A)).convert('L')
            lista = []

            gdalRaster = gdal.Open(os.path.join(str(INPUT),
str(RASTERS[selectedAindex-1])))
            #Try put A instead of gdal raster
            #QMessageBox.about(self.dlg, 'A', str(A))

            x = gdalRaster.RasterXSize
            y = gdalRaster.RasterYSize
            geo = gdalRaster.GetGeoTransform()
            minx = geo[0]
            maxy = geo[3]
            maxx = minx + geo[1] * x
            miny = maxy + geo[5] * y
            extent_raster = str(minx) + "," + str(maxx) + "," +
str(miny) + "," + str(maxy)
            pixelSize = geo[1]
```

```
            A=os.path.join(str(INPUT), str(RASTERS[selectedAindex-1]))
            B=os.path.join(str(INPUT), str(RASTERS[selectedBindex-1]))
            C=os.path.join(str(INPUT), str(RASTERS[selectedCindex-1]))
            #-1 happens because of the sapace " " index

            lista = lista + [str(A)] + [str(B)]+ [str(C)]
            dirs = ';'.join(lista)
            #QMessageBox.about(self.dlg2, 'teste', str(dirs))

            OUT=self.dlg2.lineEdit_5.text()

            Processing.initialize()

            selected_index=self.dlg2.comboBox.currentIndex()

            srcdata= self.dlg2.doubleSpinBox.value()

            out2 =
QFileInfo(QgsApplication.qgisUserDbFilePath()).path() + "/out2"

            if selected_index==0:
                pass
            # ------------------------------Color composite----------
----------------------------
            if selected_index==1:

                Processing.runAlgorithm("gdalogr:merge", None,
str(dirs), False, True, 5, str(OUT) + ".tif")

            # ----------------------NDVI-------------------------------
----------------------------
            if selected_index==2:
                #1 - Red B4 e 2 - NIR B5

#Processing.runAlgorithm("saga:rastercalculator",None,str(A),str(B),"(
b-a)/(b+a)",False,7,str(OUT)+".tif")

Processing.runAlgorithm("gdalogr:rastercalculator",None,str(A),"1",str
(B),"1",None,"1",None,"1",None,"1",\
                                        None,"1","(B-
A)/(B+A)","",5,"",str(OUT)+".tif")
            # ---------------------------------NDWI-------------------
----------------------------
            if selected_index==3:


                Processing.runAlgorithm("gdalogr:rastercalculator",
None, str(A), "1", str(B), "1", None, "1", None, "1",\
                                        None, "1",None, "1", "(B-
A)/(B+A)", "", 5, "", str(OUT) + ".tif")

            # ----------------------------EVI-----------------------
----------------------------
            if selected_index==4:
                #Use GDAL raster calculator
                Processing.runAlgorithm("gdalogr:rastercalculator",
None, str(A), "1", str(B), "1", str(C), "1", None,\
                                        "1", None, "1",None, "1",
```

```
"(2.5*(B-A))/(B+6*A-7.5*C+1)", "", 5, "", str(OUT) + ".tif")


        #Pan sharpenning-----------------------------------------------
-------------------------------------------
        Multi = self.dlg2.lineEdit_4.text()

        Pan = self.dlg2.lineEdit_6.text()

        super= self.dlg2.lineEdit_2.text()


        Out_pan = self.dlg2.lineEdit_7.text()

        if not(len(Multi)==0 and len(Pan)==0):

            Processing.initialize()
                #

Processing.runAlgorithm("otb:superimposesensor",None,str(Pan),str(Mult
i),0, 4, 0, 0, 2, 128,super)


            if len(super)==0:
                Processing.runAlgorithm("otb:pansharpeningrcs", None,
str(Pan), str(Multi), 0, 128, str(Out_pan) + ".tif")
            else:
                Processing.runAlgorithm("otb:pansharpeningrcs", None,
str(Pan), str(super), 0, 128, str(Out_pan) + ".tif")


    # ------------------------------------------------RUN3------------
-------------------------------------------------

    def run3(self):
        """Run method that performs all the real work"""
        # show the dialog
        self.dlg3.show()

        QObject.connect(self.dlg3.buttonBox, SIGNAL("accepted()"),
self.classification)

    def classification(self):

        inshapefile3 = self.dlg3.lineEdit.text()

        #INPUT Read RGB file
        RGB = self.dlg3.lineEdit_3.text()


        Out_Un = self.dlg3.lineEdit_4.text()

        Un_Cl=self.dlg3.lineEdit_2.text()

        ts_value = self.dlg3.spinBox.value()
        nc_value = self.dlg3.spinBox_2.value()
        maxit_value = self.dlg3.spinBox_3.value()
```

```
        ct_value = self.dlg3.doubleSpinBox.value()
        nan_value = self.dlg3.spinBox_4.value()


Processing.runAlgorithm("otb:unsupervisedkmeansimageclassification",No
ne,str(RGB), 128, None, ts_value, nc_value,

maxit_value,ct_value,str(Out_Un)+".tif", None)

        subprocess.call(["gdalwarp.exe", "-srcnodata", str(nan_value),
"-dstnodata", "nan", "-crop_to_cutline", "-cutline",

inshapefile3,str(Out_Un)+".tif",str(Un_Cl)+".tif"])

        QMessageBox.about(self.dlg3, 'Classification', "complete")
```